The Dissertation Committee for Sindhu Vijaya Raghavan
certifies that this is the approved version of the following dissertation:

Bayesian Logic Programs for Plan Recognition and Machine Reading

Committee:

Raymond J. Mooney, Supervisor

Kenneth Barker

Joydeep Ghosh

Pradeep Ravikumar

Jude Shavlik

Bayesian Logic Programs for Plan Recognition and Machine Reading

by

Sindhu Vijaya Raghavan, B.E.; M.S.C.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2012

To Gautam

Acknowledgments

First and foremost, I would like to thank my advisor Ray Mooney who has been a great influence on me during my PhD. I admire Ray's passion for research and science, which has often motivated me to keep going during the difficult times in my PhD. He has been sympathetic and understanding during the times when progress has been slow. I am extremely thankful to him for encouraging me to work on topics outside my dissertation, which has helped me develop diversity in my work. He has been a terrific mentor and has taught me how to do good research. Above all, he is a lot of fun to work with and it has been a great pleasure knowing him. I will miss all the weekly meetings and the group outings that were organized by the ML group.

I would also like to thank my committee members Ken Barker, Joydeep Ghosh, Pradeep Ravikumar, and Jude Shavlik for their valuable feedback on my PhD dissertation. Apart from serving on my committee, Joydeep Ghosh has mentored me on projects outside my dissertation and I would like to thank him for his effort in helping me get my work published. I am also thankful to Jude Shavlik for coming down to Austin for my defense. Despite moving to IBM, Ken Barker has been very helpful in getting me the necessary data for my PhD dissertation and I am extremely thankful to him for that.

I would like to thank all the past and present members of the Machine Learn-

ing Group at the University of Texas at Austin. I would like to especially thank Parag Singla who has been a mentor to me during my PhD. I would also like to thank Tuyen Huynh, Rohit Kate, and Yinon Bentor for the valuable discussions and feedback on my work. Finally, I would like to thank the rest of the group: Bishal Burman, David Chen, Dan Garrette, Joe Reisinger, Joohyun Kim, Ayan Acharya, Hyeonseo Ku, Lu Guo, Tanvi Motwani, and Karl Pichotta for attending my practice talks and giving me valuable feedback on my talks and presentations.

I have also had the opportunity to collaborate with several other people on topics outside my dissertation. I would like to thank Adriana Kovashka and Suriya Gunasekar for their collaboration on course projects and I appreciate their efforts in working with me closely to get our work published.

I would like to thank all the staff members at the Department of Computer Science for their help and support during my PhD. I would like to especially thank Katherine Utz, a former staff member for her help while applying to the doctoral program at UT. I would also like to thank Gloria Ramirez, Lydia Griffith, and Stacy Miller who have been incredibly helpful in sorting out various administrative issues during the course of my PhD. Finally, I would like to thank David Kotz and the Mastodon team for their efforts in maintaining the Mastodon cluster, without which I could not have run any experiments and worked on my research.

I would like to thank all my friends in Austin, especially Juhun Lee and Eunjung Choi who have made this journey a lot of fun for me. I will cherish and miss all the times I have spent with them.

I would like to profusely thank Mia Markey for her help and support during my PhD. I have known Mia Markey through my husband, Gautam Muralidhar, who happens to be Mia's PhD student. Mia has been a mentor, a source of inspiration, and a role model to me in the last few years. It has been a real pleasure knowing her and I will truly miss all the talks and discussions I have had with her.

I would like to thank the UTCS department for supporting me with the MCD fellowship in the first year of my PhD. My research was also funded by MURI ARO grant W911NF-08-1-0242 and Air Force Contract FA8750-09-C-0172 under the DARPA Machine Reading Program. Experiments were run on the Mastodon Cluster, provided by NSF grant EIA-0303609. Finally, I would like to thank the SIRE team from IBM for providing SIRE extractions on one of the data sets used in this dissertation.

I would like to thank my father, Vijayaraghavan, my mother, Premalatha and my sister, Shubha Raghavan for supporting me to pursue my PhD. I would also like to thank my father-in-law, Muralidhar and my uncle Sitaram Mirle for their encouragement. I would specially thank my cousin Kiran Mirle for his constant interest in my work, which meant a lot to me.

This PhD would not have been possible without the encouragement and support of my loving husband, Gautam Muralidhar. He has been my source of strength and inspiration throughout this journey. He has been by my side during the good times and the bad times. I am extremely thankful to him for making this journey a memorable one. Also, the birth of our son, Vivan, who at the time of writing this is a month old has made this journey a very special one for both of us.

SINDHU VIJAYA RAGHAVAN

The University of Texas at Austin

December 2012

Bayesian Logic Programs for Plan Recognition and Machine Reading

Publication No. _____

Sindhu Vijaya Raghavan, Ph.D.
The University of Texas at Austin, 2012

Supervisor: Raymond J. Mooney

Several real world tasks involve data that is uncertain and relational in nature. Traditional approaches like first-order logic and probabilistic models either deal with structured data or uncertainty, but not both. To address these limitations, statistical relational learning (SRL), a new area in machine learning integrating both first-order logic and probabilistic graphical models, has emerged in the recent past. The advantage of SRL models is that they can handle both uncertainty and structured/relational data. As a result, they are widely used in domains like social network analysis, biological data analysis, and natural language processing. Bayesian Logic Programs (BLPs), which integrate both first-order logic and Bayesian networks are a powerful SRL formalism developed in the recent past. In this dissertation, we develop approaches using BLPs to solve two real world tasks – plan recognition and machine reading.

Plan recognition is the task of predicting an agent's top-level plans based on its observed actions. It is an abductive reasoning task that involves inferring cause

from effect. In the first part of the dissertation, we develop an approach to abductive plan recognition using BLPs. Since BLPs employ logical deduction to construct the networks, they cannot be used effectively for abductive plan recognition as is. Therefore, we extend BLPs to use logical abduction to construct Bayesian networks and call the resulting model Bayesian Abductive Logic Programs (BALPs).

In the second part of the dissertation, we apply BLPs to the task of machine reading, which involves automatic extraction of knowledge from natural language text. Most information extraction (IE) systems identify facts that are explicitly stated in text. However, much of the information conveyed in text must be inferred from what is explicitly stated since easily inferable facts are rarely mentioned. Human readers naturally use common sense knowledge and "read between the lines" to infer such implicit information from the explicitly stated facts. Since IE systems do not have access to common sense knowledge, they cannot perform deeper reasoning to infer implicitly stated facts. Here, we first develop an approach using BLPs to infer implicitly stated facts from natural language text. It involves learning uncertain common sense knowledge in the form of probabilistic first-order rules by mining a large corpus of automatically extracted facts using an existing rule learner. These rules are then used to derive additional facts from extracted information using BLP inference. We then develop an online rule learner that handles the concise, incomplete nature of natural-language text and learns first-order rules from noisy IE extractions. Finally, we develop a novel approach to calculate the weights of the rules using a curated lexical ontology like WordNet.

Both tasks described above involve inference and learning from partially

observed or incomplete data. In plan recognition, the underlying cause or the top-level plan that resulted in the observed actions is not known or observed. Further, only a subset of the executed actions can be observed by the plan recognition system resulting in partially observed data. Similarly, in machine reading, since some information is implicitly stated, they are rarely observed in the data. In this dissertation, we demonstrate the efficacy of BLPs for inference and learning from incomplete data. Experimental comparison on various benchmark data sets on both tasks demonstrate the superior performance of BLPs over state-of-the-art methods.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Several real world tasks involve data that is uncertain and relational in nature. Traditional approaches like first-order logic and probabilistic models either deal with structured data or uncertainty, but not both. To address these limitations, statistical relational learning (SRL) (Getoor & Taskar, 2007), a new area in machine learning integrating both first-order logic and probabilistic graphical models has emerged in the recent past. The advantage of SRL models is that they can handle both uncertainty and structured/relational data.

Let us consider the example of predicting the task that a user is performing on a computer based on the actions performed by the user. The user could be performing the task of copying a file or moving a file from one directory to another, and he could be working on several different files at the same time. The files and directories represent different entities and the tasks the user is performing represent different relations between those entities. Now, the prediction task involves inferring not only the correct relation, but also the entities that participate in the relation. Purely statistical learning techniques like Bayesian networks or Markov networks cannot be used for such problems as these models are essentially propositional in nature. Even though purely first-order logic based approaches handle structured

data, they cannot handle uncertainty. But there is always uncertainty in real data - uncertainty in the relations between different entities, uncertainty in the types of entities, etc. By combining strengths of both first-order logic and statistical models, SRL formalisms lend themselves to solving such real world tasks effectively.

Because of their advantages, SRL formalisms are widely used for social network analysis (e.g. (Richardson & Domingos, 2006)), biological data analysis (e.g. (Perlich & Merugu, 2005; Huynh & Mooney, 2008)), information extraction (e.g. (Bunescu & Mooney, 2007)), and other domains that involve structured/relational data. As a result, the last few years have seen a development of several SRL formalisms like Probabilistic Relational Models (PRMs) (Friedman, Getoor, Koller, & Pfeffer, 1999), Stochastic Logic Programs (SLPs) (Muggleton, 2000), Bayesian Logic Programs (BLPs) (Kersting & De Raedt, 2001, 2007), Markov Logic Networks (MLNs) (Richardson & Domingos, 2006) etc. BLPs, which integrate first-order logic and Bayesian networks are a simple, yet powerful formalism for solving problems with structured data. One advantage of BLPs over other SRL formalisms like MLNs is with regard to the grounding process used to construct Bayesian networks. Unlike MLNs, BLPs do not include all possible groundings of a rule in the ground network. Instead, they include only those groundings of the rules that are used to deduce or prove the query. As a result, the ground networks constructed by BLPs are much smaller than those constructed by models like MLNs, enabling BLPs to scale to large domains. Further, due to the directed nature of BLPs, it is possible to use any type of logical inference to construct the networks. As we will see in Chapter 3 in this dissertation, even though BLPs use logical deduction

by default to construct ground networks, it is also possible to employ logical abduction to construct ground Bayesian networks. As a result, they can be used to solve tasks involving abductive as well as deductive reasoning efficiently. Finally, since Bayesian networks are a mature technology, a lot of existing machinery developed for Bayesian networks like algorithms for probabilistic inference can be used for BLPs as well. Because of these reasons, we have chosen to use BLPs in our research.

In this dissertation, we develop approaches using BLPs to solve two real world tasks – plan recognition and machine reading. Plan recognition involves inferring an intelligent agent's top-level plans based on its observed actions. It has practical applications in several domains including monitoring activities of daily living for elderly care, intelligent surveillance systems, and intelligent personal assistants or user interfaces. Machine reading involves automatic extraction of information from natural language text. Like plan recognition, machine reading is also widely used in practical applications such as deep question answering. For these reasons, we have focused on developing approaches using BLPs on these two tasks.

In the first part of the dissertation, we develop an approach to abductive plan recognition using BLPs. Plan recognition is an abductive reasoning task that involves inferring cause from effect (Charniak & McDermott, 1985). The example of a prediction task described above is an instance of plan recognition in intelligent user interfaces. Since BLPs employ logical deduction to construct the networks, they cannot be used effectively for abductive plan recognition as is. Therefore, we extend BLPs to use logical abduction to construct Bayesian networks and call the

resulting model Bayesian Abductive Logic Programs (BALPs).

In the second part of the dissertation, we develop approaches to machine reading using BLPs. Most information extraction (IE) systems (Cowie & Lehnert, 1996; Sarawagi, 2008) identify facts that are explicitly stated in text. However, much of the information conveyed in text must be inferred from what is explicitly stated since easily inferable facts are rarely mentioned. Human readers naturally use common sense knowledge and "read between the lines" to infer such implicit information from the explicitly stated facts. Since IE systems do not have access to common sense knowledge, they cannot perform deeper reasoning to infer implicit facts. Consider the text "Barack Obama is the president of the United States of America." Given the query "Barack Obama is a citizen of what country?", standard IE systems cannot identify the answer since citizenship is not explicitly stated in the text. However, a human reader possesses the common sense knowledge that the president of a country is almost always a citizen of that country, and easily infers the correct answer.

To this end, we first develop an approach using BLPs to infer implicit facts from natural language text. It involves learning uncertain common sense knowledge in the form of probabilistic first-order rules by mining a large corpus of automatically extracted facts using an existing rule learner. These rules are then used to derive additional facts from explicitly stated information using BLP inference. We then develop an online rule learner that handles the concise, incomplete nature of natural-language text and learns first-order rules from noisy IE extractions. Finally, we develop a novel approach to calculate the weights of the rules using a curated

lexical ontology like WordNet (Fellbaum, 1998).

Both tasks described above involve inference and learning from partially observed or incomplete data. In plan recognition, the underlying cause or the top-level plan that resulted in the observed actions is not known or observed. Further, only a subset of the executed actions can be observed by the plan recognition system resulting in partially observed data. Similarly, in machine reading, since some information is implicit, they are rarely observed in the data. In this dissertation, we demonstrate the efficacy of BLPs for inference and learning from incomplete data. Experimental comparison on various benchmark data sets on both tasks demonstrate the superior performance of BLPs over state-of-the-art methods.

1.1 Dissertation Contributions

The first contribution involves extending BLPs for abductive plan recognition. BLPs use Selective Linear Definite clause (SLD) resolution to generate proof trees, which are then used to construct a ground Bayesian network for a given query. However, deduction is unable to construct proofs for abductive problems such as plan recognition because deductive inference involves predicting effects from causes, while the plan recognition task involves inferring causes (top-level plans) from effects (observations). Therefore, we extend BLPs to use logical abduction to construct proofs. In logical abduction, missing facts are assumed when necessary to complete proof trees, and we use the resulting abductive proof trees to construct Bayesian networks. We call the resulting model Bayesian Abductive Logic Programs (BALPs) (Raghavan & Mooney, 2011). We learn the parameters

for the BALP framework automatically from data using the Expectation Maximization algorithm adapted for BLPs by Kersting and De Raedt (2008). Experimental evaluation on three benchmark data sets demonstrate that BALPs outperform the existing state-of-art methods like MLNs for plan recognition.

The second contribution involves developing an approach that uses BLPs to infer implicit facts from natural language text for machine reading. Our approach involves learning common sense knowledge in the form of probabilistic first-order rules by mining a substantial database of facts that an IE system has already automatically extracted from a large corpus of text and subsequently using those rules to deduce additional information from the extracted facts using the BLP framework. Due to the concise and incomplete nature of natural language text, facts that are easily inferred from explicitly stated facts are rarely mentioned in the text. As a result, the main challenge in this task involves learning first-order rules from a few instances of inferable facts seen in the training data. For the same reason, the facts extracted by an IE system are always quite noisy and incomplete, as a result of which a purely logical approach to learning and inference is unlikely to be effective. Hence, we learn probabilistic first-order rules using LIME (McCreath & Sharma, 1998), an existing rule learner that is capable of handling noisy training data and then use the resulting BLP to make effective probabilistic inferences when interpreting new documents (Raghavan, Mooney, & Ku, 2012). Experimental evaluation of our system on a realistic test corpus from DARPA's Machine Reading project demonstrates improved performance compared to a purely logical approach based on Inductive Logic Programming (ILP) (Lavrač & Džeroski, 1994), and an

alternative SRL approach based on MLNs.

The final contribution in this dissertation involves developing a novel online rule learner that is capable of learning first-order rules from noisy and incomplete natural language extractions, which are then used to infer implicit facts from natural language text for machine reading. Most existing inference-rule learners (Quinlan, 1990; McCreath & Sharma, 1998; Srinivasan, 2001; Kersting & De Raedt, 2008) assume that the training data is largely accurate, and hence are not adept at learning useful rules from noisy and incomplete IE output. Also, most of them do not scale to large corpora. Due to these limitations, we have developed an efficient online rule learner that handles the concise, incomplete nature of natural-language text by learning rules in which the body of the rule typically consists of relations that are frequently explicitly stated, while the head is a relation that is more typically inferred. We use the frequency of occurrence of extracted relations as a heuristic for distinguishing those that are typically explicitly stated from the ones that are usually inferred. In order to allow scaling to large corpora, we develop an efficient online rule learner. Experimental evaluation on the machine reading task demonstrates superior performance of our rule learner when compared to LIME, an existing rule learner used in our previous approach.

As an additional contribution, we also develop a novel approach to scoring first-order rules learned from IE extractions for the purpose of inferring implicit facts from natural language text for machine reading. Probabilistic inference using the BLP framework requires learning parameters (conditional probability table (CPT) entries) for the learned rules. Since those relations that are easily inferred

from explicitly stated facts are seldom seen in the training data, learning useful parameters using conventional parameter learning approaches like EM for BLPs (Kersting & De Raedt, 2008) have resulted in limited success (Raghavan et al., 2012). Consequently, we develop an alternate approach to specifying parameters for the learned first-order rules using lexical information from a curated ontology like WordNet (Fellbaum, 1998). The basic idea behind our approach is that more accurate rules typically have predicates that are closely related to each other in terms of the meanings of the English words used to name them. Since WordNet is a rich resource for lexical information, we use it for scoring rules based on word similarity. Experimental evaluation on the machine reading task demonstrates superior performance of the our approach over both manual weights and weights learned using EM.

1.2 Dissertation Outline

The remainder of the dissertation is organized as follows:

- Chapter 2 reviews terminology and notation used in this dissertation. It also reviews background on logical abduction, inductive logic programming, Bayesian networks, and BLPs.

- Chapter 3 describes our approach to abductive plan recognition using BLPs. Experimental evaluation of our BLP based approach on three benchmark datasets from plan recognition demonstrates its efficacy and superior performance over existing state-of-the-art approaches including MLNs.

- Chapter 4 describes our approach to inferring implicit facts from natural language text using BLPs. Experimental evaluation of our resulting system on a realistic test corpus from DARPA's Machine Reading project demonstrates improved performance compared to a purely logical approach based on ILP, and an alternative SRL approach based on MLNs.

- Chapter 5 describes our online rule learner for learning first-order rules from noisy and incomplete natural language extractions. We also describe our approach to scoring the learned rules using WordNet. Experimental evaluation on the test corpus from DARPA's Machine Reading project demonstrates the efficacy of the proposed methods.

- Chapter 6 discusses future work and chapter 7 concludes the dissertation.

We note that the material presented in Chapter 3 has appeared in our previous publication Raghavan and Mooney (2011) and the material presented in Chapter 4 has appeared in the publication Raghavan et al. (2012).

Chapter 2

Background

2.1 First-order logic

First-order logic is a formal language for representing relational domains involving several objects, their properties, and their relationships with other objects (Russell & Norvig, 2003). A *term* in first-order logic is a symbol that represents an object or an entity in the domain and every object in the domain has an associated type. There are three types of terms – constants, variables, and functions. A *constant* is a term that represents an individual object or an entity, while a *variable* is a term that acts as a template or a placeholder for a set of entities of the same type. A *function symbol*, represented by f/n is a term that represents a function over a set of terms; f is the name of the function symbol, and n is the arity, or the number of arguments/terms it takes. All constants are represented using strings that start with a lower–case letter (e.g *mary*, *bob*, *alice*), while all variables are represented using strings that start with an upper–case (e.g X1, Y1, Z1). A *predicate*, denoted by p/n represents a relation between entities in the domain; p is the name of the predicate, and n is its arity, the number of arguments/terms the predicate takes.

A literal is a predicate applied to terms. A positive literal is called an atom, and a negative literal is a negated atom. A literal that contains only constants is

called a ground literal. A ground atom whose truth value is known is called a *fact*. A clause is an expression of the form

$$b_1 \wedge b_2 \wedge .... \wedge b_n \rightarrow h_1 \vee h_2 \vee ..... h_n$$

where '$\wedge$' represents a conjunction, '$\vee$' represents a disjunction, and '$\rightarrow$' stands for an implication. $b_1 \wedge b_2 \wedge .... \wedge b_n$ is called the *body* or *antecedent* of the clause, while $h_1 \vee h_2 \vee ..... h_n$ is called the *head* or *consequent* of the clause. The above clause can also be written in the disjunctive normal form (DNF) as a disjunction of literals as follows:

$$\neg b_1 \vee \neg b_2 \vee .... \vee \neg b_n \vee h_1 \vee h_2 \vee ..... h_n$$

A *Horn clause* is a clause in DNF that contains at most one positive literal and a *definite clause* is one that contains exactly one positive literal. If b_i and h are atoms, then a definite clause has the form

$$b_1 \wedge b_2 \wedge .... \wedge b_n \rightarrow h$$

Given a logical formula, the variables in the formula are either universally quantified or existentially quantified. A variable is said to be universally quantified if it is true for all objects in the domain. On the other hand, a variable is said to be existentially quantified if it is true for some object in the domain. The symbol

'$\forall$' is used for universal quantification, while the symbol '$\exists$' is used for existential quantification.

A substitution $\theta = \{V_1/t_1, V_2/t_2, ...V_n/t_n\}$ is an assignment of terms t_i to corresponding variables V_i. Given a formula (term, atom, clause) f and a substitution θ, then the instantiation $f\theta$ represents the formula obtained by replacing each variable V_i in the formula by its corresponding term t_i in θ. *Unification* is the process that takes two atomic sentences p and q and returns a substitution θ such that $p\theta = q\theta$, if both of them match, otherwise it returns failure.

Given a logic program, i.e. a set of first-order clauses, the *Herbrand universe* is defined as the set of all ground terms that can be constructed from the constants and function symbols that are present in the program. For a function free logic program, the Herbrand universe reduces to the set of constants that occur in the clauses. The *Herbrand base* is the set of ground atoms over the Herbrand universe. The *Herbrand interpretation* is the set of ground atoms from the Herbrand base that are true. A Herbrand interpretation I is a model for a clause c if and only if for all substitutions θ such that $body(c)\theta \subset I \rightarrow head(c)\theta \in I$. I is a model for a set of clauses B if it is a model for every clause in B.

Automated inference in first-order logic involves using techniques like forward chaining and backward chaining. Given a knowledge base (KB) consisting of a set of formulae in first-order logic and a set of facts, forward chaining adds new facts to the knowledge base. For every implication $p \rightarrow q$ in the KB, if p is satisfied, i.e. if p is true, then q is added to the KB, if it is not already present. On the other hand, given a KB and a query literal, backward chaining searches for those impli-

cations which can derive the query literal. For a query literal q, if an implication $p \rightarrow q$ is present, and if p is true, then backward chaining concludes that q is true, otherwise it will try to prove p. If p cannot be proved, then it fails to conclude q. *SLD resolution* is a backward chaining procedure for definite clauses.

2.2 Logical Abduction

Abduction, also called abductive reasoning, is defined as the process of finding the best explanation for a set of observations (Peirce, 1958). It is widely used in tasks such as plan/activity recognition and diagnosis that require inferring cause from effect (Ng & Mooney, 1992). Most previous approaches to abduction have been based on first-order logic and determine a small set of assumptions sufficient to deduce the observations (Pople, 1973; Levesque, 1989; Kakas, Kowalski, & Toni, 1993). In the logical framework, abduction is usually defined as follows (Pople, 1973):

- **Given:** Background knowledge B and observations O, both represented as sets of formulae in first-order logic, where B is typically restricted to a set of definite clauses and O is restricted to a conjunction of ground literals.
- **Find:** A hypothesis H, also a set of logical formulae, such that $B \cup H \not\models \bot$ and $B \cup H \models O$.

Here $\models$ means logical entailment and $\bot$ means false, i.e. find a set of assumptions that is consistent with the background theory and explains the observations. There

are generally many hypotheses H that explain a particular set of observations O. The best hypothesis is typically selected based on the size (simplicity) of H, following Occam's Razor.

2.3 Inductive Logic Programming

Inductive logic programming (ILP) has been defined as the intersection of machine learning and logic programming (Muggleton, 1992). Given background knowledge B and a set of positive and negative examples for a target relation/predicate, ILP involves finding a hypothesis H, usually a definite logic program such that H along with B covers most of the positive examples, but none of the negative examples. However, to account for the presence of noise in the data, most systems allow a few negative examples to be covered. The background knowledge B can either be a definite logic program or a set of ground literals that represent entities and relations in the domain. De Raedt and Kersting (2004) discuss the following settings for ILP systems :

- Learning from entailment:

  In this setting, the induced hypothesis H, along with the background knowledge B entails all positive examples, but does not entail any of the negative examples. FOIL (Quinlan & Cameron-Jones, 1993), ALEPH (Srinivasan, 2001), PROGOL (Muggleton, 1995), and LIME (McCreath & Sharma, 1998) are some of the ILP systems that are learn from entailment.

- Learning from interpretations:

In this setting, examples represent Herbrand interpretations and the induced hypothesis H is said to cover an example if and only if the example is a Herbrand model of $B \cup H$, where B is the background knowledge base. CLAUDIEN (De Raedt & Dehaspe, 1997) is an ILP system that learns from interpretations. Learning from interpretations is easier than learning from entailment as the examples in the former are complete, i.e. all literals that are true are known. Further, this setting is suitable for learning in domains where only positive examples are available.

- Learning from proofs:

  In this setting, examples are ground proof-trees and the induced hypothesis H is said to cover an example, if and only if the example is a proof of $B \cup H$, where B is the background knowledge base. Model Inference System (MIS) (Shapiro, 1983) is a system that learns from proofs.

2.3.1 LIME

In this dissertation, we use LIME to learn first-order rules in Chapter 4. LIME is an ILP system that learns complete candidate hypotheses (a set of clauses) instead of candidate clauses. Instead of using a greedy set covering approach that is used in systems like FOIL, LIME uses a Bayesian heuristic to search the space of possible hypotheses. Unlike systems like FOIL that handle noise by allowing some negative examples to be covered, LIME uses a generative model that incorporates an explicit parameter to handle noise. As a result of a more sophisticated model, LIME has a superior ability to handle noise when compared to other ILP systems

like FOIL and PROGOL. Further, LIME has the added ability to learn from only positive, only negative, and both positive and negative examples. Due to these advantages, we chose to use LIME to learn first-order rules in this dissertation.

2.4 Bayesian Networks

A Bayesian network is a directed acyclic graph that represents the joint probability distribution of a set of random variables in a compact manner (Koller & Friedman, 2009). Each node in the network represents a random variable and the directed edges between nodes represent the conditional dependencies between the random variables. If there is a directed edge from node a to node b, then the random variable represented by node b is conditionally dependent on that represented by node a. Absence of edges between nodes indicate conditional independence between the random variables. In a discrete Bayesian network, each node takes a discrete set of values. Associated with each node is a conditional probability table (CPT), which gives the probability of the node taking a certain value for different combination of values that the parent nodes take. The joint probability distribution for a Bayesian network is given by the following formula:

$$\text{P}(X) = \prod_i P(X_i|Pa(X_i)),$$

where $X = X_1, X_2, ..., X_n$ represents the set of random variables in the network and $Pa(X_i)$ represents the parents of X_i. For the rest of this dissertation, we will discuss only discrete Bayesian networks.

2.4.1 Probabilistic Inference in Bayesian Networks

Inference in Bayesian networks generally involves computing the posterior probability of a query given the evidence. Koller and Friedman (2009) describe several algorithms to perform both exact and approximate inference in discrete Bayesian networks. The junction tree algorithm is one of the most commonly used exact inference methods for Bayesian networks. All exact inference methods including the junction tree algorithm have a computational complexity that is exponential in size of the maximal clique in the network. As a result, for networks that are densely connected, exact inference is usually intractable. When exact inference is intractable, approximate inference techniques are used for probabilistic inference.

For approximate inference, several sampling algorithms like forward sampling, likelihood weighting, Gibbs sampling etc. can be used. These algorithms generate a large number of random samples for the given Bayesian network, and then use these samples to compute posterior probabilities. However, if the underlying Bayesian network contains several deterministic constraints, i.e. 0 values in the CPTs, then these sampling algorithms fail to generate sufficient samples, thus leading to a poor approximation of the posterior probability. In such cases, SampleSearch (Gogate & Dechter, 2007), an approximate sampling algorithm specifically designed for graphical models with multiple deterministic constraints can be used.

The other type of inference in Bayesian networks involves computing the most probable explanation (MPE) (Pearl, 1988), which determines the joint assignment of values to unobserved nodes in the network that results in maximum

posterior probability given the evidence. It is possible to compute multiple alternative explanations using the k-MPE algorithm (Nilsson, 1998). There are several Bayesian network packages like Netica[1] and ELVIRA (Elvira-Consortium, 2002) that provide implementation for some of these algorithms. We use Netica for exact inference (joint and marginal), ELVIRA for MPE and k-MPE inference, and SampleSearch for approximate inference.

2.4.2 Learning Bayesian Networks

Learning Bayesian networks automatically from data involves learning the structure, i.e. the conditional dependencies between the random variables, and learning the parameters, i.e. the entries in the CPTs. Given a Bayesian network with a fixed structure, it is possible to learn the parameters automatically from data. When the data is fully observable, frequency counting is used to estimate the maximum likelihood (ML) parameters (Koller & Friedman, 2009). For partially observed data, Lauritzen (1995) has proposed an algorithm based on Expectation Maximization (EM), which maximizes the likelihood of the data. Russell et al. (1995) have proposed a gradient-ascent based parameter learning algorithm that also optimizes the likelihood of the data. On the other hand, Greiner and Zhou (2002) have developed a method for discriminatively learning the parameters by optimizing the conditional likelihood. Several methods have been proposed to learn the structure of Bayesian networks automatically from data (Koller & Friedman, 2009). Some methods use dynamic programming and its extensions to learn the

[1] `http://www.norsys.com/`

structure (Koivisto & Sood, 2004; Silander & Myllymäki, 2006). Other methods learn the structure approximately by searching through the space of possible network structures (Heckerman & Chickering, 1995) or searching through the space of possible network orderings (Teyssier & Koller, 2005).

2.4.3 Noisy-Or and Noisy-And models

The noisy-or and noisy-and models (Pearl, 1988) are used to encode the CPTs for Boolean nodes compactly using fewer parameters. In a discrete Bayesian network, the number of entries in the CPTs for any node is exponential in the number of parents. However, the noisy-or and noisy-and models require parameters that are linear in the number of parents for encoding the CPT. As a result, these models help reduce the number of parameters that have to be estimated from data.

The *noisy-or model* is used when there are several different causes c_i for an event e and each cause independently triggers the event with a certain probability p_i. Let $leak$ be the probability that the event e occurs due to an unknown cause. Then the probability of occurrence of e using the noisy-or is given as below:

$$P(e) = 1 - [(1 - leak)] \prod_i (1 - p_i)^{c_i}$$

The *noisy-and* model is used where there are several events c_i that have to occur simultaneously for the event e to occur and each event c_i fails to trigger e independently with a probability p_i. Let inh be the probability that e does not occur even when all events c_i have occurred. inh accounts for unknown events due to

which e has failed to trigger. Then the probability of occurrence of e using the noisy-and model is as below:

$$P(e) = (1 - inh) \prod_i (1 - p_i)^{(1-c_i)}$$

2.5 Bayesian Logic Programs

Bayesian logic programs (BLPs) (Kersting & De Raedt, 2001, 2007) can be considered as templates for constructing *directed* graphical models (Bayesian networks). Given a knowledge base as a special kind of logic program, standard logical inference (SLD resolution) is used to automatically construct a Bayesian network for a given problem. More specifically, given a set of facts and a query, all possible definite-clause proofs of the query are constructed and used to build a Bayesian network for answering the query. Standard probabilistic inference techniques described in Section 2.4.1 are then used to compute the most probable answer.

More formally, a BLP consists of a set of *Bayesian clauses*, definite clauses of the form $a|a_1, a_2, a_3, ...a_n$, where $n \geq 0$ and a, a_1, a_2, a_3,.....,a_n are *Bayesian predicates* (defined below). a is called the head of the clause (head(c)) and (a_1, a_2, a_3,....,a_n) is the body (body(c)). When $n = 0$, a Bayesian clause is a fact. Each Bayesian clause c is assumed to be universally quantified and range restricted, i.e. $variables\{head\} \subseteq variables\{body\}$. If the head has variables that are not present in the body, then these variables cannot always be bound to constants during deductive inference. Each Bayesian clause has an associated *conditional probability distribution* cpd(c) = P(head(c)|body(c)), also referred to as conditional probability

table (CPT) in this dissertation.

A *Bayesian predicate* is a predicate with a finite domain, and each ground atom for a Bayesian predicate represents a random variable. Associated with each Bayesian predicate is a combining rule such as *noisy-or* or *noisy-and* that maps a finite set of cpds into a single cpd (cf. Section 2.4.3). Let a be a Bayesian predicate defined by two Bayesian clauses, $a|a_1, a_2, a_3, ...a_n$ and $a|b_1, b_2, b_3, ...b_n$, where cpd_1 and cpd_2 are their cpd's. Let θ be a substitution that satisfies both clauses. Then, in the constructed Bayesian network, directed edges are added from the nodes for each $a_i\theta$ and $b_i\theta$ to the node for $a\theta$. The combining rule for a is used to construct a single cpd for $a\theta$ from cpd_1 and cpd_2. Note that if there is no θ that satisfies multiple clauses, then no combining rule is used to combine the evidence coming from the two clauses. The probability of a joint assignment of truth values to the final set of ground propositions is then defined in the standard way for a Bayesian network:

$$\text{P}(X) = \prod_i P(X_i|Pa(X_i)),$$

where $X = X_1, X_2, ..., X_n$ represents the set of random variables in the network and $Pa(X_i)$ represents the parents of X_i. Once a ground network is constructed, standard probabilistic inference methods can be used to answer various types of queries as described in Section 2.4.1. In this dissertation, we use the deterministic logical-and combining rule to combine evidence from the conjuncts in the body of the clause. To combine evidence coming from different rules that have the same

head, we use the noisy-or model, whose parameters are learned using the methods described in Section 2.5.1. We use the noisy-or model since it is the standard approach for encoding a cpd to support "explaining away". Explaining away refers to the phenomenon that evidence for one explanation decreases confidence in alternative competing explanations (Pearl, 1988). We use exact probabilistic inference (marginal and joint) as implemented in Netica[2] a commercial Bayes-net software package when possible. When exact inference is not tractable, we use SampleSearch (Gogate & Dechter, 2007), an approximate sampling algorithm specifically designed for graphical models with multiple deterministic constraints. In our models, the deterministic constraints, i.e. the 0 values in the cpds arise due to the use of the logical-and model to combine the evidence coming from the literals in the body of the clauses.

2.5.1 Learning Bayesian Logic Programs

Learning a BLP from data involves learning the structure, the set of first-order definite clauses and the parameters – the cpd entries and the parameters for the combining rules. Given a BLP with a fixed structure, the parameters of the BLP can be learned automatically from data using the methods proposed by Kersting and De Raedt (2008). In their first method, Kersting and De Raedt have adapted the Expectation Maximization (EM) algorithm developed for propositional Bayesian networks by Lauritzen (1995), and in their second method, they have adapted the gradient-ascent based algorithm developed by Russell et al. (1995) for Bayesian

[2] `http://www.norsys.com/`

networks. Note that both the algorithms proposed by Kersting and De Raedt for learning the parameters of a BLP optimize the likelihood of the data. Kersting and De Raedt have also proposed an algorithm to learn the structure of the BLP based on the model of learning from interpretations (Kersting & De Raedt, 2008). In their method, each example in the training data represents the least Herbrand model of the target BLP. The algorithm performs a hill climbing search through the space of possible structures by optimizing the likelihood of the data. They use CLAUDIEN (De Raedt & Dehaspe, 1997) to get the initial set of structures for the search.

Parameter learning in BLPs using EM

In this dissertation, we learn parameters for the BLP framework using the EM algorithm, adapted for BLPs by Kersting and De Raedt (2008). The EM algorithm is designed to learn the cpd entries and parameters for the combining rules from incomplete data. Typically, if the values for all variables are seen during training, the maximum likelihood estimation of the parameters reduces to frequency counting. In the presence of incomplete data, i.e. if values for some random variable are not seen during training, the EM algorithm is used. Similar to the EM algorithm for propositional Bayesian networks, EM for BLPs learns the parameters by optimizing the likelihood of the observed data.

Given the structure of the BLP (first-order definite clauses) and a set of training instances, where each instance is a set of literals observed in the data, the EM algorithm for BLPs works as described below. For each training instance, the BLP inference performs SLD resolution given the observed literals and constructs a ground Bayesian network. Next, in the expectation step, also called the E step,

expected counts are computed for all parameters in the model. The E step is similar to that for propositional Bayesian networks where the expected counts represent a distribution of possible completions over the incomplete data. In the case of BLPs, since a first-order rule could be grounded in several different ways, each grounding is treated as an independent example or instance of the first-order clause while estimating expected counts. In the next step, called the maximization step or the M step, the algorithm computes maximum likelihood estimates using the expected counts. Note that in the fully observed data case, the maximum likelihood estimates are computed using the actual counts instead of expected counts.

EM starts by initializing all parameters to random values or user defined values. It converges to a local optimum when the estimated values for the parameters remain constant across successive iterations. Since the EM is a greedy hill climbing algorithm, it often gets stuck in a local optimum. Random restart is a popularly used approach to get out of local optima. Kersting and De Raedt (2008) adapt the EM algorithm to estimate parameters for combining rules by using only decomposable rules to combine evidence from multiple rules, since such rules allow for the parameters to be estimated independently of other parameters. A rule is said to be decomposable if every node in the ground network is derived using exactly one clause (Kersting & De Raedt, 2008).

Chapter 3

Plan Recognition using Bayesian Logic Programs

3.1 Introduction

In this chapter, we describe our approach to abductive plan recognition using Bayesian Logic Programs (BLPs). Plan recognition is the task of predicting an agent's top-level plans based on its observed actions. It is an abductive reasoning task that involves inferring cause from effect (Charniak & McDermott, 1985). It is used in several applications including monitoring activities of daily living for elderly care, story understanding, strategic planning, intelligent user interfaces, and intelligent personal assistants. In this chapter, we apply plan recognition to story understanding, strategic planning, and intelligent user interfaces. In story understanding, the character's motives or plans have to be recognized based on its actions in order to answer questions about the story. In strategic planning systems where there are several agents performing several actions, it becomes necessary for each agent to recognize plans of other agents so that they can work cooperatively. In an intelligent user interface, plan recognition is used to predict the task the user is performing so that the system could give valuable tips to the user to perform the task more efficiently.

Traditionally, plan-recognition approaches have been based on first-order

logic in which a knowledge-base of plans and actions is developed for the domain and then default reasoning (Kautz & Allen, 1986) or logical abduction (Ng & Mooney, 1992) is used to predict the best plan based on the observed actions. However, these approaches are unable to handle uncertainty in the observations or background knowledge and are incapable of estimating the likelihood of different plans. An alternative approach to plan recognition is to use probabilistic methods such as Abstract Hidden Markov Models (Bui, 2003), probabilistic context-free grammars (Pynadath & Wellman, 2000), Bayesian networks (Charniak & Goldman, 1989, 1991; Huber, Durfee, & Wellman, 1994; Horvitz & Paek, 1999), or statistical n-gram models (Blaylock & Allen, 2005b). While these approaches handle uncertainty, they cannot handle structured representations as they are essentially propositional in nature. As a result, it is also difficult to incorporate planning domain knowledge in these approaches.

As mentioned earlier, the last few years have seen a development of several SRL formalisms such as MLNs and BLPs. Of these formalisms, MLNs have been applied to abductive plan recognition by Kate and Mooney (2009). Since MLNs employ deduction for logical inference, they adapt MLNs for abduction by adding reverse implications for every rule in the knowledge base. However, the addition of these rules increases the size and complexity of the MLN, resulting in a computationally expensive model. We refer to this MLN model by Kate and Mooney as MLN-PC, where PC stands for "pairwise constraints". In order to overcome the limitations of MLN-PC, Singla and Mooney (2011) have developed two new approaches to plan recognition using MLNs. In the first approach which we refer to

as MLN-HC (here HC stands for "hidden cause"), they extend MLN-PC by adding a hidden cause for each rule antecedent, which reduces the complexity of ground networks to some extent. However, the MLN-HC model still results in complex networks, which prevents it from scaling to large domains. In the second approach which we refer to as MLN-HCAM (here HCAM stands for "hidden cause abductive model construction"), they enhance the MLN-HC model by incorporating a novel model construction procedure based on logical abduction that results in much simpler and smaller networks, thereby making this approach scale to large domains. MLN-HCAM model uses the logical abduction procedure used in our BLP based approach to construct ground networks in MLNs. Based on the success of our BLP based approach, Singla and Mooney (2011) explored the possibility of using logical abduction to further constrain the size of the ground networks created by the MLN-HC model. MLN-HCAM outperforms both MLN-HC and MLN-PC.

In this chapter, we describe our approach to abductive plan recognition using BLPs. Pearl (1988) argued that causal relationships and abductive reasoning from effect to cause are best captured using directed graphical models (Bayesian networks). Since plan recognition is abductive in nature, we believe that a directed probabilistic logic like BLPs will be better suited for plan recognition than an undirected probabilistic logic like MLNs. Further, since BLPs include only those groundings of the rules that are used to prove the query in the ground network, we hypothesize that BLPs can overcome some of the limitations of MLN-PC (Kate & Mooney, 2009) and MLN-HC (Singla & Mooney, 2011) with respect to scaling to large domains.

As described earlier, BLPs use SLD resolution to generate proof trees, which are then used to construct a ground Bayesian network for a given query. Since SLD resolution is a deductive inference, it is unable to construct proofs for abductive problems such as plan recognition because deductive inference involves predicting effects from causes, while the plan recognition task involves inferring causes (top-level plans) from effects (observations). Therefore, we extend BLPs to use logical abduction to construct proofs. In logical abduction, missing facts are assumed when necessary to complete proof trees, and we use the resulting abductive proof trees to construct Bayesian networks. We call the resulting model Bayesian Abductive Logic Programs (BALPs). Like all SRL formalisms, BALPs combine the strengths of both first-order logic and probabilistic graphical models, thereby overcoming the limitations of traditional plan recognition approaches mentioned above.

The rest of this chapter is organized as follows. We first describe the abductive inference procedure used in BALPs in Section 3.2. In Section 3.3, we describe how probabilistic parameters are specified and how probabilistic inference is performed. We discuss how parameters can be automatically learned from data in Section 3.4. In Section 3.5, we describe our experimental evaluation of BALPs on three plan recognition data sets from three different application domains – story understanding, strategic planning, and intelligent user interfaces. In Section 3.6, we describe related work and summarize in Section 3.7.

3.2 Abductive Inference in BALPs

Let O_1, O_2, ..., O_n be the set of observations. We derive a set of most-specific abductive proof trees for these observations using the method originally proposed by Stickel (1988). The abductive proofs for each observation literal are computed by backchaining on each O_i until every literal in the proof is proven or assumed. A literal is said to be proven if it unifies with some fact or the head of some rule in the knowledge base, otherwise it is said to be assumed. Since multiple plans/actions could generate the same observation, an observation literal could unify with the head of multiple rules in the knowledge base. For such a literal, we compute alternative abductive proofs. The resulting abductive proof trees are then used to build the structure of the Bayesian network using the standard approach for BLPs.

The basic algorithm to construct abductive proofs is given in Algorithm 1. The algorithm takes as input a knowledge base (KB) in the form of definite clauses and a set of observations as ground facts. It outputs a set of abductive proof trees by performing logical abduction on the observations. These proof trees are then used to construct the Bayesian network. For each observation O_i, AbductionBALP searches for rules whose consequents (heads) unify with O_i. For each such rule, it computes the substitution from the unification process and substitutes variables in the body of the rule with bindings from the substitution. The literals in the body now become new subgoals in the inference process. If these new subgoals cannot be proved, i.e. if they cannot unify with existing facts or with the consequent of any rule in the KB, then they are assumed. In order to minimize the number of

Algorithm 1 AbductionBALP

Inputs: Background knowledge KB and observations O_1, O_2, O_3, ..., O_n both represented as sets of formulae in first-order logic, where KB is typically restricted to a set of definite clauses and each O_i is a ground literal.

Output: Abductive proofs for all O_i.

```
1: Let Q be a queue of unproven atoms, initialized with O_i
2: while Q not empty do
3:    A_i ← Remove atom from Q
4:    for each rule R_i in KB do
5:       consequent ← Head literal of R_i
6:       if A_i unifies with consequent then
7:          S_i ← unify A_i and consequent and return substitution
8:          Replace variables in the body of R_i with bindings in S_i. Each literal in
            the body of R_i is a new subgoal.
9:          for each literal_i in body of R_i do
10:            if literal_i unifies with head of some rule R_j in KB then
11:               add literal_i to Q
12:            else if literal_i unifies with an existing fact then
13:               Unify and consider the literal to be proved
14:            else
15:               if literal_i unifies with an existing assumption then
16:                  Unify and update the assumption
17:               else
18:                  Assume literal_i by replacing any unbound variables that are ex-
                     istentially quantified in literal_i with new Skolem constants.
19:               end if
20:            end if
21:         end for
22:      end if
23:   end for
24: end while
```

assumptions, the assumed literals are first matched with existing assumptions. If no such assumption exists, then any unbound variables in the literal that are existentially quantified are replaced by Skolem constants. We note that there are no assumptions or facts to match at the start. We also note that since the observation literals are ground literals, they are never matched to any assumptions of facts in the knowledge base.

In SLD resolution, which is used in BLPs, if any subgoal literal cannot be proven, the proof fails. However, in BALPs, we assume such literals and allow proofs to proceed till completion. Note that there could be multiple existing assumptions that could unify with subgoals in Step 15. However, if we used all ground assumptions that could unify with a literal, then the size of the ground network would grow exponentially, making probabilistic inference intractable. In order to limit the size of the ground network, we unify subgoals with assumptions in a greedy manner. We found that this approach worked well for the plan-recognition domains we explored. For other tasks, domain-specific heuristics could potentially be used to reduce the size of the network.

We now illustrate the abductive inference process with a simple example from the Story-Understanding benchmark data set described in Section 3.5.1. Consider the partial knowledge base and set of observations given in Figure 3.1a and Figure 3.1b respectively. There are two top-level plans, shopping and robbing, in the knowledge base. Note that the action literal *inst(G, going)* could be observed as part of both shopping and robbing. For each observation literal in Figure 3.1b, we recursively backchain to generate abductive proof trees. When we backchain

(a)
Shopping
1. inst(G,going) | inst(B,shopping), go-step(B,G).
2. inst(SP,shopping-place) | inst(S,shopping), store(S,SP).
Robbing
3. inst(P,going) | inst(R,robbing), go-step(R,P).

(b)
inst(go1,going)
inst(store1,shopping-place)

(c)
inst(go1,going) | inst(a1,shopping), go-step(a1,go1).
inst(go1,going) | inst(a1,robbing), go-step(a1,go1).
inst(store1,shopping-place) | inst(a1,shopping), store(a1,store1).

Figure 3.1: (a) A partial knowledge base from the Story Understanding data set. (b) The logical representation of the observations. (c) The set of ground rules obtained from logical abduction.

on the literal *inst(go1,going)* using Rule 1, we obtain the subgoals *inst(B,shopping)* and *go-step(B,go1)*. These subgoals become assumptions since no observations or heads of clauses unify with them. Since B is an existentially quantified variable, we replace it with a Skolem constant $a1$ to obtain the ground assumptions *inst(a1,shopping)* and *go-step(a1,go1)*. We then backchain on literal *inst(go1,going)* using Rule 3 to get subgoals *inst(R,robbing)* and *go-step(R,go1)*. We cannot unify *inst(R, robbing)* with any observation or existing assumptions; however, we can unify *go-step(R,go1)* with an existing assumption *go-step(a1,go1)*, thereby binding R to $a1$. In order to minimize the number of assumptions, we first try to match literals with unbound variables to existing assumptions, rather than instantiating them with new Skolem constants. Finally, we backchain on the literal *inst(store1,shopping-*

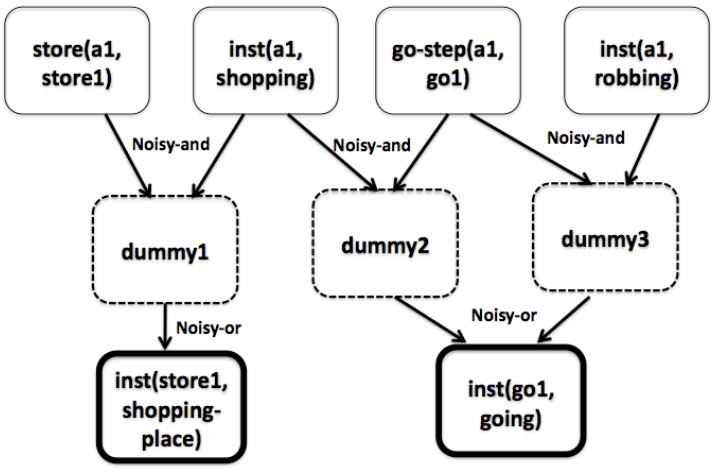


Figure 3.2: Bayesian network constructed for example in Figure 3.1. The nodes with thick borders represent observed actions, the nodes with dotted borders represent intermediate nodes used to combine the conjuncts in the body of a clause, and the nodes with thin borders represent plan literals.

place) using Rule 2 to get subgoals *inst(S,shopping)*, *store(S,store1)*. Here again, we match *inst(S, shopping)* to an existing assumption *inst(a1,shopping)*, thereby binding S to $a1$.

Figure 3.1c gives the final set of ground rules generated by abductive inference. After generating all abductive proofs for all observation literals, we construct a Bayesian network. Figure 3.2 shows the Bayesian network constructed for the example in Figure 3.1. Note that since there are no observations/facts that unify with the subgoals (*inst(B,shopping)*, *go-step(B,G)*, *inst(R,robbing)*, *go-step(R,P)*, and *store(S,SP)*) generated during backchaining on observations, SLD resolution will fail to generate proofs. This is typical in plan recognition, and as a result, we cannot use BLPs for such tasks.

The only difference between BALPs and BLPs lies in the logical inference procedure used to construct proofs. Once the abductive proofs are generated, BALPs use the same procedure as BLPs to construct the Bayesian network. We further show in Section 3.4 and Section 3.5.3 that techniques developed for BLPs for learning parameters can also be used for BALPs.

3.3 Probabilistic Parameters and Inference

We now discuss how parameters are specified in BALPs. We use noisy/logical-and and noisy-or models to specify the cpds in the ground Bayesian network as these models compactly encode the cpd with fewer parameters, i.e. just one parameter for each parent node. Depending on the domain, we use either a strict *logical-and* or a softer *noisy-and* model to specify the cpd for combining evidence from the conjuncts in the body of a clause. We use a noisy-or model to specify the cpd for combining the disjunctive contributions from different ground clauses with the same head. Figure 3.2 shows the noisy-and and noisy-or nodes in the Bayesian network constructed for the example in Figure 3.1.

Given the constructed Bayesian network and a set of observations, we determine the best explanation using standard methods for computing the Most Probable Explanation (MPE) (see Chapter 2), which determines the joint assignment of values to the unobserved nodes in the network that has the maximum posterior probability given the observations. To compute multiple alternative explanations, we use the k-MPE algorithm (Nilsson, 1998) as implemented in Elvira (Elvira-Consortium, 2002). For other types of exact probabilistic inference (marginal and joint) we use

Netica,[1] a commercial Bayes-net software package.

When the complexity of the ground network makes exact inference intractable (as in the Monroe dataset described in Sect. 3.5), we have to resort to approximate inference. Due to the (noisy/logical) *and* and *or* nodes in the network, there are a number of deterministic constraints, i.e. 0 values in the cpds. As a result, generic importance sampling algorithms like likelihood weighting failed to generate sufficient samples. Hence, we used SampleSearch (Gogate & Dechter, 2007), an approximate sampling algorithm specifically designed for graphical models with multiple deterministic constraints.

3.4 Parameter Learning

Learning can be used to automatically set the noisy-or and noisy-and parameters in the model. We learn these parameters using the EM algorithm adapted for BLPs by Kersting and De Raedt (2008) (see Chapter 2). In supervised training data for plan recognition, one typically has evidence for the observed actions and the top-level plans. However, we usually do not have evidence for network nodes corresponding to subgoals, noisy-ors, and noisy/logical-ands. As a result, there are a number of variables in the ground networks which are always hidden, and hence EM is appropriate for learning the requisite parameters from the partially observed training data. We simplify the problem by learning only the noisy-or parameters and using a deterministic logical-and model to combine evidence from the conjuncts in

[1] `http://www.norsys.com/`

the body of a clause. We use uniform priors for top-level plans unless otherwise mentioned.

3.5 Experimental Evaluation

In this section, we evaluate BALPs on three plan-recognition datasets. Unfortunately, there are very few benchmark datasets or rigorous experimental evaluations of plan recognition. First, we describe experiments to demonstrate that BALPs are more effective for plan recognition than previous approaches. Then, we describe additional experiments to demonstrate that the EM algorithm can learn parameters of the BALP model effectively.

3.5.1 Datasets

3.5.1.1 Monroe / Reformulated Monroe

We used the Monroe dataset, an artificially-generated plan-recognition dataset in the emergency response domain by Blaylock and Allen (2005a). This domain includes top level plans such as setting up a temporary shelter, clearing a road wreck, and providing medical attention to victims. The task is to infer a single top level plan from a set of observed actions automatically generated by a planner. The planner used to construct plans is SHOP2 (Nau, Ilghami, Kuter, Murdock, Wu, & Yaman, 2003) and the domain knowledge is represented as a hierarchical transition network (HTN). We constructed a logical knowledge base consisting of 153 clauses to represent the domain knowledge encoded in the HTN. We used 1,000 artificially generated examples in our experiments. Each example instantiates one of 10 top-

Top-level plan template	Description
set-up-shelter(Loc)	Set up an emergency shelter at location Loc
fix-water-main(From,To)	Fix a power line running from From to To
clear-road-hazard(From,To)	Clear a road hazard from From to To
plow-road (From,To)	Plow road from From to To
fix-power-line(Loc)	Fix power line at location Loc
provide-medical-attention(Person)	Provide medical attention to person Person

Table 3.1: Templates for a subset of top-level plans from the Monroe data set

Observed action template
navegate-snowplow(Driver, Plow, From)
engage-plow(Driver, Plow)
navegate-vehicle(Person, Veh, Loc)
climb-in(Person, Veh)
climb-out(Person, Veh)
treat(Emt, Person)

Table 3.2: Templates for a subset of observed actions from the Monroe data set

level plans and contains an average of 10.19 literals describing a sample execution of this plan. The total number of action predicates in this data set is 30. The templates for a subset of top-level plans and observed actions are given in Table 3.1 and Table 3.2 respectively. This data set is an example of plan recognition being applied to strategic planning. The knowledge base constructed for Monroe can be found in Appendix A.

Due to computational complexity, we were unable to compare the performance of BALPs with Kate and Mooney's MLN-PC (2009) approach on this domain. Their approach resulted in an MLN with rules containing multiple existentially quantified variables which produced an exponential number of possible

groundings, eventually leading to memory overflow. In order to compare BALPs with MLN-PC, we slightly modified the Monroe domain to eliminate this problem without significantly changing the underlying task. The resulting dataset also had 1,000 examples, with an average of 9.7 observations per example. We refer to this dataset as "Reformulated-Monroe." The knowledge base constructed for Reformulated-Monroe can be found in Appendix A.

3.5.1.2 Linux

The Linux dataset is another plan-recognition dataset created by Blaylock and Allen (2004). Human users were asked to perform various tasks in Linux and their commands were recorded. The task is to predict the correct top level plan from the sequence of executed commands. For example, one of the tasks involves finding all files with a given extension. The dataset consists of 19 top level plans and 457 examples, with an average of 6.1 command literals per example. The total number of action predicates in this domain is 43. The templates for a subset of top-level plans and observed actions are given in Table 3.3 and Table 3.4 respectively. We constructed the background knowledge base consisting of 50 clauses for the Linux dataset based on our knowledge of the commands. The knowledge base constructed for Linux can be found in Appendix A. This data set is an example of plan recognition being applied to intelligent user interfaces.

Top-level plan template	Description
find-file-by-attr-name-ext(Ext)	Find a file that has the extension Ext
know-filespace-usage-file(File)	Find how much space the file File uses
determine-machine-connected-alive(Mac-name)	Find if the machine Mac-name is up
create-file(File,Dir)	Create a file File in the directory Dir
remove-files-by-attr-name-ext(Ext)	Remove all files with the extension Ext

Table 3.3: Templates for a subset of top-level plans from the Linux data set

Observed action template
mv(Dest-prepath, Dir, Source-prepath, Name)
cp(Dest-prepath, Dir, Source-prepath, Name)
find(Prename, Name, Size, Prepath, Path)
cd(Prepath, Path)
ls(Path, Name)
ping(Machine-name, Machine-path)
vi(Prepath, Filename)
mkdir(Dir, Parent-dir-name)

Table 3.4: Templates for a subset of observed actions from the Linux data set

3.5.1.3 Story Understanding

We also used a dataset[2] that was previously used to evaluate abductive story understanding systems (Ng & Mooney, 1992; Charniak & Goldman, 1991). In this task, characters' higher-level plans must be inferred from their actions described in a narrative text. A logical representation of the literal meaning of the text is given for each example. A sample story is: "Bill went to the liquor-store. He pointed a gun at the owner." There are 12 top-level plans in this dataset including shopping, robbing, restaurant dining, traveling in a vehicle (bus, taxi or plane), partying and jogging. Some narratives involve more than a single plan. This small dataset consists of 25 development examples and 25 test examples each containing an average of 12.6 literals. We used the background knowledge that was initially constructed for the ACCEL system (Ng & Mooney, 1992). This data set is an example of plan recognition being applied to story understanding. Figure 3.1a and Figure 3.1b in Section 3.2 give a sample knowledge base and a set of observations from the Story Understanding data set.

Apart from the fact that each data set comes from a different application domain, all data sets described above test certain specific aspects of the plan recognition system. Since the Monroe domain is very large with several subgoals and entities, it tests the ability of the plan recognition system to scale to large domains. On the other hand, the Linux data set does not have a large domain. However, since the data is from human users, it is noisy. There are several sources of noise includ-

[2] `http://www.cs.utexas.edu/~ml/accel.html`

ing the case in which the human users have reported that they have successfully executed the top-level plan even though they have not (Blaylock & Allen, 2005b). As a result, this data set tests the robustness of the plan recognition system. The plan recognition task on Monroe and Linux domains involve prediction of a *single* top-level plan based on the observed actions. However, on the Story Understanding domain, most examples have multiple top-level plans as the answer. This data set tests the ability of a plan recognition system to identify all possible top-level plans based on the observed actions.

3.5.2 Comparison with Other Approaches

We now present comparisons to previous approaches to plan recognition across different benchmark datasets.

3.5.2.1 Monroe and Linux

We first compared BALPs with MLN-HCAM (Singla & Mooney, 2011) and Blaylock and Allen's (2005b) plan-recognition system on both the Monroe and Linux datasets. Blaylock and Allen's approach learns statistical n-gram models to separately predict plan schemas (i.e. predicates) and their arguments. We were unable to run MLN-PC and MLN-HC on these domains due to scaling issues.

We learn the noisy-or parameters for BALPs using the EM algorithm described in Sect. 3.4 for both Linux and Monroe domains. We initially set all noisy-or parameters to 0.9 and this gave reasonable performance in both domains. We ran EM with several different starting points including random weights and manual

weights (0.9). We found that running EM starting with manual weights generally performed the best for both domains, and hence we used the weights learned from this model for comparison.

For Linux, we performed 10-fold cross validation for evaluation and we ran EM till convergence on the training set for each fold. For Monroe, where more data is available, we used 300 examples for training, 200 examples for validation, and the remaining 500 examples for testing. We ran EM iterations on the training set until the accuracy on the validation set stopped improving. We then used the final learned set of weights to perform plan-recognition on the test set.

For both Monroe and Linux, the plan-recognition task involves inferring a *single* top level plan that best explains the observations. Hence, we computed the marginal probabilities for all ground instantiations of the plan predicates in the network and picked the single plan instantiation with the highest marginal probability.

Due to differences in Blaylock and Allen's experimental methodology and ours, we are only able to compare performance directly using the convergence score (Blaylock & Allen, 2005b). The convergence score is the fraction of examples for which the correct plan predicate is inferred (ignoring the arguments) when given *all* of the observations.

Table 3.5 shows the results. BALPs outperform both MLN-HCAM and Blaylock and Allen's system on the convergence score in both domains[3]. The convergence scores for MLN-HCAM and Blaylock and Allen's system on Monroe are

[3]Since convergence scores for individual examples were not available for Blaylock and Allen's system, we could not perform the test for statistical significance.

	BALPs	MLN-HCAM	Blaylock and Allen
Convergence-Monroe	**98.4**%	97.0%	94.2%
Convergence-Linux	**46.6**%	38.9%	36.1%

Table 3.5: Results for BALPs, MLN-HCAM, and the system by Blaylock and Allen on Monroe and Linux.

already quite high, leaving little room for improvement. However, BALPs were still able to improve over MLN-HCAM by 1.44% and Blaylock and Allen's system by 4.45%. On the other hand, the baseline convergence scores for Linux were fairly low, and BALPs were able to improve over MLN-HCAM by 19.67% and Blaylock and Allen's system by a remarkable 29.1%. Despite this improvement, the overall convergence score for Linux is not that high. Noise in the data is one reason for the modest score. Another issue with this data set is the presence of very similar plans, like find-file-by-ext and find-file-by-name. The commands executed by users in these two plans are nearly identical, making it difficult for a plan recognition system to distinguish them (Blaylock & Allen, 2004).

Partial Observability Results

The convergence score has the following limitations as a metric for evaluating the performance of plan recognition:

1. It only accounts for predicting the correct plan predicate, ignoring the arguments. In most domains, it is important for a plan-recognition system to predict arguments accurately as well. For example, in the Linux domain, if the user is trying to move "test1.txt" to "test-dir", it is not sufficient to predict

the move command; it is also important to predict the file (test.txt) and the destination directory (test-dir).

2. It only evaluates plan prediction after the system has observed *all* of the executed actions. However, in most cases, we would like to be able to predict plans after observing as few actions as possible.

In order to evaluate the ability of BALPs to infer plan arguments and to predict plans after observing only a partial execution, we conducted an additional set of experiments. Specifically, we performed plan recognition after observing the first 25%, 50%, 75%, and 100% of the executed actions. To measure performance, we compared the complete inferred plan (with arguments) to the gold-standard to compute an overall *accuracy* score. When computing accuracy, partial credit was given for predicting the correct plan predicate with only a subset of its correct arguments. A point was rewarded for inferring the correct plan predicate, then, given the correct predicate, an additional point was rewarded for each correct argument. For example, if the correct plan was $plan_1(a_1, a_2)$ and the inferred plan was $plan_1(a_1, a_3)$, the accuracy was 66.67%.

We compare BALPs with MLN-HCAM to measure their ability to perform plan recognition on partially observable data. Blaylock and Allen did not perform these experiments, and hence we do not compare BALPs performance to that of their system on partially observable data. Table 3.6 and Table 3.7 show the results for partial observability on Monroe and Linux respectively. On Monroe, BALPs perform slightly better than MLN-HCAM on higher levels of observability, whereas

	25%	50%	75%	100%
BALPs	07.33	20.26	44.63	**79.16**
MLN-HCAM	15.93	19.93	43.93	76.30

Table 3.6: Accuracy on Monroe at varying levels of observability

	25%	50%	75%	100%
BALPs	**19.83**	**25.45**	**34.06**	**36.32**
MLN-HCAM	16.30	16.48	24.36	28.84

Table 3.7: Accuracy on Linux at varying levels of observability

MLN-HCAM tends to outperform BALP on lower levels of observability. However, on Linux BALPs outperform MLN-HCAM at all levels of partial observability.

3.5.2.2 Reformulated-Monroe

We also compared the performance of BALPs with Kate and Mooney's (2009) MLN-PC approach on the Reformulated-Monroe dataset[4]. For MLN-PC, we were unable to learn clause weights effectively on this dataset since it was intractable to run Alchemy's[5] existing weight-learners due to the sizes of the MLN and data. Hence, we manually set the weights using the heuristics described by Kate and Mooney (2009). To ensure a fair comparison, we use manual weights for BALPs instead of using learned weights; we uniformly set all noisy-or parameters to .9 and used logical-and to combine the evidence from the conjuncts as this model gave reasonable performance on other domains as well.

[4]We were unable to compare to MLN-HC and MLN-HCAM (Singla & Mooney, 2011) as the results were not available on this data set.

[5]`http://alchemy.cs.washington.edu/`

	25%	50%	75%	100%	Convergence Score
BALPs	**9.83**	**32.67**	**66.80**	**97.40**	**99.90**
MLN-PC	4.10*	19.26*	40.51*	79.20*	79.66*

Table 3.8: Comparative Results for Reformulated-Monroe. "*" indicates that the differences in the performance are statistically significant.

We used both the convergence score and accuracy to compare the performance of the two approaches. Similar to Monroe and Linux, the observation set for this domain includes all actions executed to achieve the top level plan. In order to evaluate performance for partially observed plans, we performed plan recognition after observing the first 25%, 50%, 75%, and 100% of the executed actions.

Table 3.8 shows the results. BALPs consistently outperform the MLN approach on this data set and the differences in their performance are statistically significant as determined by the Wilcoxon Sign Rank (WSR) test (Rosner, 2005). Significance was concluded at the 0.05 level. The convergence score for BALPs improved over that for MLN-PC by a significant 25.41%.

3.5.2.3 Story Understanding

On Story Understanding, we compared the performance of BALPs to MLN-HC (Singla & Mooney, 2011), MLN-HCAM (Singla & Mooney, 2011), MLN-PC (Kate & Mooney, 2009) and ACCEL (Ng & Mooney, 1992), a purely logic-based system that uses a metric to guide its search for selecting the best explanation. ACCEL can use two different metrics: *simplicity*, which selects the explanation with the fewest assumptions and *coherence*, which selects the explanation that maxi-

mally connects the input observations. This second metric is specifically geared towards text interpretation by measuring *explanatory coherence* (Ng & Mooney, 1990). Currently, this bias has not been incorporated in either the BALP or any of the MLN approaches.

For BALPs, we were unable to learn useful parameters from just 25 development examples. As a result, we set parameters manually trying to maximize performance on the development set. As before, a uniform value of 0.9 for all noisy-or parameters seemed to work well for this domain. Unlike other domains, using the logical-and model to combine the evidence from conjuncts in the body of the clause did not yield good results on Story Understanding. However, we found that using the noisy-and model improved the results by a fair margin; so we set the noisy-and parameters to a uniform value of 0.9. However, to disambiguate between conflicting plans, we set different priors for high level plans to maximize performance on the development data.

The explanations generated by different MLN approaches include additional facts implied by the minimal explanation. To compare fairly different systems, we constructed high level plans from the predicted and answer ground literals using the rules shown in Figure 3.3. We describe the construction of high level plans using the plan *plan-shopping(smarket-shopping,Person1,Thing1,Place1)* as an example. An instance of this plan is constructed with *smarket-shopping* as the first argument when an instance of *inst(S,smarket-shopping)* is inferred or is present in the answer set. For the remaining arguments in the high level plan, appropriate constants are used when the respective literals are inferred or are present in the answer

set, otherwise NULL is used. Since multiple plans are possible in this domain, we compared the inferred plans with the ground truth to compute *precision*, *recall*, and *F-measure*, the harmonic mean of precision and recall. As before, partial credit was given for predicting the correct plan predicate with some incorrect arguments. The observed literals in this data are already incomplete and do not include all of the actions needed to execute a plan, so they were used as is.

Table 3.9 shows the results. As before, an "*" indicates that the difference in the performances of a given method and that of BALPs is statistically significant as determined by the Wilcoxon Sign Rank (WSR) test (Rosner, 2005) using a significance level of 0.05. BALPs performed better than ACCEL-Simplicity (ACCEL-Sim) and the different MLN based approaches. With respect to F-measure, BALPS improved over MLN-HCAM by 8.52%, MLN-HC by 7.87%, MLN-PC by 15.57%, and ACCEL-Simplicity by a significant 33.65%. However, ACCEL-Coherence (ACCEL-Coh) still performed the best. Since the coherence metric incorporates extra criteria specific to story understanding, this bias would need to be included in the probabilistic models to make them more competitive. However, the coherence metric is specific to narrative interpretation and not applicable to plan recognition in general.

Overall, we found that BALPs outperformed most existing approaches on the benchmark data sets, thus demonstrating that BALPs are effective for plan recognition.

```
plan-shopping(smarket-shopping,Person1,Thing1,Place1)|
inst(S,smarket-shopping),shopper(S,Person1),thing-shopped-for(S,Thing1),
store(S,Place1).

plan-shopping(liqst-shopping,Person1,Thing1,Place1)|
inst(S,liqst-shopping),shopper(S,Person1),thing-shopped-for(S,Thing1),
store(S,Place1).

plan-shopping(shopping,Person1,Thing1,Place1)|
inst(S,shopping),shopper(S,Person1),thing-shopped-for(S,Thing1),
store(S,Place1).

plan-robbing(robbing,Person1,Place1,Victim1,Weapon1,Thing1)|
inst(R,robbing),robber(R,Person1),place-rob(R,Place1),
victim-rob(R,Victim1),weapon-rob(R,Weapon1),thing-robbed(R,Thing1).

plan-air-travel(going-by-plane,Person1,Luggage1,Place1,Tkt1,Plane1)|
inst(P,going-by-plane),goer(P,Person1),plane-luggage(P,Luggage1),
source-go(P,Place1),plane-ticket(P,Tkt1),vehicle(P,Plane1).

plan-bus-travel(going-by-bus,Person1,Bus1,Source1,Dest1,Driver1,Tkn1)|
inst(B,going-by-bus),goer(B,Person1),vehicle(B,Bus1),
source-go(B,Source1),dest-go(B,Dest1),bus-driver(B,Driver1),token(B,Tkn1).

plan-rest-dining(rest-dining,Person1,Rest1,Thing1,Drink1,Instrument1)|
inst(D,rest-dining),diner(D,Person1),restaurant(D,Rest1),
rest-thing-ordered(D,Thing1),rest-thing-drunk(D,Drink1),
rest-drink-straw(D,Instrument1).

plan-drinking(drinking,Person1,Drink1,Instrument1)|
inst(D,drinking),drinker(D,Person1),patient-drink(D,Drink1),
instr-drink(D,Instrument1).

plan-taxi-travel(going-by-taxi,Person1,Taxi1,Source1,Dest1,Td1)|
inst(B,going-by-taxi),goer(B,Person1),vehicle(B,Taxi1),source-go(B,Source1),
dest-go(B,Dest1),taxi-driver(B,Td1).

plan-paying(paying,Person1,Thing1)|
inst(P,paying),payer(P,Person1),thing-paid(P,Thing1).

plan-jogging(jogging,Person1,Drink1,Instrument1)|
inst(J,jogging),jogger(J,Person1),jog-thing-drunk(J,Drink1),
jog-drink-straw(J,Instrument1).

plan-partying(partying,Person1,Drink1,Instrument1)|
inst(P,partying),agent-party(P,Person1),party-thing-drunk(P,Drink1),
party-drink-straw(P,Instrument1).
```

Figure 3.3: Rules used to construct high level plans for Story Understanding

	BALP	MLN-HCAM	MLN-HC	MLN-PC	ACCEL-Sim	ACCEL-Coh
Precision (%)	72.07	69.13	67.08	67.31	66.45	89.39*
Recall (%)	85.57	75.32*	78.94*	68.10*	52.32*	89.39
F-measure (%)	78.24	72.10	72.53	67.70*	58.54*	89.39*

Table 3.9: Comparative Results for Story Understanding. "*" indicates that the differences wrt BALPs are statistically significant.

3.5.3 Parameter Learning Experiments

We now describe additional experiments that we performed to understand the EM algorithm for learning the parameters for BALPs. These experiments are designed to demonstrate that the EM algorithm is effective for learning the parameters for BALPs on different plan recognition domains.

3.5.3.1 Learning Methodology

We used EM as described in Sect. 3.4 to learn noisy-or parameters for the Linux and Monroe domains[6]. We initially set all noisy-or parameters to 0.9. This gives reasonable performance in both domains, so we compare BALPs with learned noisy-or parameters to this default model which we call "Manual-Weights" (MW). For training, we ran EM with two sets of starting parameters – manual weights (0.9) and random parameters. We call the former "MW-Start" and the latter "Rand-Start". We used the same training and test splits as described above for Linux and Monroe domains. To measure performance, we computed convergence score and accuracy score for various levels of observability as described above.

[6]We were unable to learn useful parameters for Story Understanding since the mere 25 development examples were insufficient for training.

	25%	50%	75%	100%	Convergence Score
MW	18.34	21.84	28.22	30.41	39.82
MW-Start	**19.83***	**25.45***	**34.06***	**36.32***	**46.60***
Rand-Start	14.55*	20.53	29.10	31.40	41.57

Table 3.10: Results for parameter learning on Linux. "*" indicates that the differences wrt MW model are statistically significant.

3.5.3.2 Learning Results

Table 3.10 shows the results for different models on Linux. An "*" indicates that the difference in the performance scores for MW and the given model is statistically significant as determined by the Wilcoxon Sign Rank (WSR) test using a significance level of 0.05. MW-Start consistently outperforms MW, demonstrating that parameter learning improves the performance of default BALP parameters on the Linux domain. Rand-Start does marginally better than MW for all but 50% and 25% levels of partial observability. However, it does not perform as well as MW-Start, showing that learning from scratch is somewhat better than using default parameters but not as effective as starting learning from reasonable default values.

Table 3.11 shows the results for different models on Monroe. The performance of MW is already so high that there is little room for improvement, at least with respect to the convergence score. As a result, the MW-Start model could not improve substantially over the MW model. The manual parameters seem to be at a (local) optimum, preventing EM from making further improvements on this data. Rand-Start is performing about as well, sometimes better and sometimes worse than

	25%	50%	75%	100%	Convergence Score
MW	7.20	20.67	46.06	79.16	98.4
MW-Start	7.33	20.26	44.63*	79.16	98.4
Rand-Start	10.46*	19.7*	44.73*	79.86*	98.4

Table 3.11: Results for parameter learning on Monroe. "*" indicates that the differences wrt MW model are statistically significant.

MW, demonstrating that starting from random values the system can learn weights that are about as effective as manual weights for this domain. One reason for the high performance of the MW model on Monroe is the lack of ambiguity in the observations, i.e. there are few observed actions that are part of more than one possible plan. Overall, EM was able to automatically learn effective parameters for BALPs.

3.5.4 Comparison of BALPs to other SRL models

BLPs, BALPs, and MLNs are all languages for flexibly and compactly representing large, complex probabilistic graphical models. An alternative approach to SRL is to add a stochastic element to the deductive process of a logic program. ProbLog (Kimmig, Santos Costa, Rocha, Demoen, & De Raedt, 2008), is the most recent and well-developed of these approaches. ProbLog can be seen as extending and subsuming several previous models, such as Poole's Horn Abduction (PHA) (Poole, 1993) and PRISM (Sato, 1995). Finally, there is publicly-available implementation of ProbLog [7] that exploits the latest inference techniques based on *binary decision diagrams* (BDDs) to provide scalability and efficiency. Therefore, we at-

[7] `http://dtai.cs.kuleuven.be/problog/`

tempted to compare the performance of BALPs to ProbLog as well.

It was relatively straightforward to develop a ProbLog program for plan-recognition by appropriately formulating the planning KB used for BALPs. However, our preliminary explorations with ProbLog revealed a serious limitation that prevented us from actually performing an experimental comparison on our plan recognition datasets. In a number of the planning axioms in our KBs, existentially quantified variables occur in the body of a clause which do not occur in the head. Representing these clauses in ProbLog requires binding such variables to all possible type-consistent constants in the domain. However, this results in the ProLog inference engine attempting to construct an intractable number of explanations (i.e. proofs) due to the combinatorial number of possible combinations of these introduced constants. Therefore, it was intractable to run ProbLog on our datasets, preventing an empirical comparison. BALPs use a greedy abductive-proof construction method described in Section 3.2 to prevent this combinatorial explosion. Therefore, we believe ProbLog would need a new approximate inference algorithm for this situation in order to be practically useful for plan recognition.

Abductive Stochastic Logic Programs (ASLPs) (Chen, Muggleton, & Santos, 2008) are another SRL model that uses stochastic deduction and supports logical abduction and, therefore, could potentially be applied to plan recognition. However, we are unaware of a publicly-available implementation of ASLPs that could be easily used for experimental comparisons.

3.5.5 Discussion

We now discuss various aspects of BALPs that have led to its superior performance over existing methods. As mentioned earlier, MLN-PC (Kate & Mooney, 2009) and MLN-HC (Singla & Mooney, 2011) approaches cannot be applied to large domains like Monroe since the addition of reverse implications results in a computationally expensive model. As opposed to the explosive grounding of rules in MLN-PC and MLN-HC, BALPs use logical abduction in which only those groundings of the rules that are used to prove the query are included in the ground network. This results in networks that are much smaller in size, thus enabling BALPs to scale to large domains. Like BALPs, MLN-HCAM (Singla & Mooney, 2011) also uses logical abduction to reduce the size of the ground networks. Instead of constructing ground Markov networks directly from the abductive proofs like in BALPs, MLN-HCAM supplies these proofs to the normal grounding process in MLNs in order to construct the ground networks. The abductive proofs from logical abduction help to limit the explosive grounding process in MLN-HCAM to a certain extent, but not completely. As a result, BALPs outperform MLN-HCAM as well. Further, the use of logical abduction allows BALPs to use an existing knowledge base that was created for planning without any modification.

When Blaylock and Allen (2005b) perform instantiated plan recognition, it is done in a pipeline of two separate steps. The first step predicts the plan schema and the second step predicts the arguments given the schema. Unlike their approach, BALPs are able to jointly predict both the plan and its arguments simultaneously. We believe that BALP's ability to perform joint prediction of plans and their argu-

ments is at least partly responsible for its superior performance. In both BALP and MLN systems for plan recognition, the domain knowledge is encoded in the knowledge base, while the system by Blaylock and Allen has no access to the domain knowledge. We believe that the ability of BALPs to incorporate domain knowledge is also responsible for its superior performance.

Blaylock and Allen's system (2005b) uses 4500 examples to learn reasonable parameters on the Monroe domain. MLN-PC and MLN-HC approaches do not even scale on the Monroe domain. On the other hand, BALPs use only 300 examples for learning parameters on the Monroe domain, proving that the EM algorithm can effectively learn parameters when given reasonable number of examples. Further, the use of planning knowledge provides additional supervision to the learning algorithm, which results in more efficient learning as well. Except for the Story Understanding data set, the EM algorithm used in BALPs could learn parameters automatically from data. The inability of the EM algorithm to learn parameters on this data set could be attributed to the lack of sufficient examples more than anything. As per Kate and Mooney (2009), even the MLN-PC approach could not learn reasonable weights on the Story Understanding data set due to lack of sufficient examples. Note that it is possible to learn parameters for Reformulated-Monroe using EM, but we did not deliberately learn these parameters to ensure fair comparison with MLNs. Overall, the success of the EM algorithm for learning parameters of BALPs on the original Monroe and Linux domains demonstrates that our approach allows for automatic learning of parameters from data. As a result, our approach does not require any manual setting or tuning of parameters.

Overall, we find that our approach to plan recognition using BALPs is very effective. Our results demonstrate that BALPs outperform most existing approaches on all benchmark data sets. As mentioned earlier, each data set in our evaluation tests a specific aspect of the system. BALP's superior performance on all benchmark data sets demonstrates that BALPs are a robust system for plan recognition.

3.6 Related Work

Some of the early work in plan recognition was done by Kautz and Allen (1986, 1987). They use deductive inference to predict plans using observed actions, an action taxonomy, and a set of commonsense rules or constraints. Lesh and Etzioni's approach (1995) to goal recognition constructs a graph of goals, actions, and their schemas and prunes the network until the plans present in the network are consistent with the observed goals. The approach by Hong (2001) also constructs a "goal graph" and analyses the graph to identify goals consistent with observed actions. None of these approaches can disambiguate between competing goals and plans using probabilistic reasoning.

There are several approaches to plan recognition using Bayesian networks (Charniak & Goldman, 1989, 1991; Huber et al., 1994). Based on the observed actions and a knowledge base constructed for planning, these approaches automatically construct Bayesian networks using different heuristics. Their work is similar to BALPs, but special purpose procedures are used to construct the necessary ground networks rather than using a general-purpose probabilistic predicate logic like MLNs or BLPs. Horvitz and Paek (1999) develop an approach that uses

Bayesian networks to recognize goals in an automated conversation system; however their approach does not handle relational data.

Pynadath and Wellman (2000) extend probabilistic context-free grammars to plan recognition; Kaminka et al. (2002) developed an approach to multiagent plan recognition using dynamic Bayesian networks to perform monitoring in distributed systems; Bui et al. (2002, 2003) use Abstract Hidden Markov Models for hierarchical goal recognition; Saria and Mahadevan (2004) extend work by Bui to multiagent plan recognition systems; Albrecht et al. (1998) develop an approach based on dynamic Bayesian networks to predict plans in an adventure game; however, none of these approaches can handle relational data. We have already discussed the other systems for plan recognition (Ng & Mooney, 1992; Kate & Mooney, 2009; Blaylock & Allen, 2005b) in Section 3.5.2.

Poole (1993) has developed a framework for Horn clause abduction using Bayesian networks. Chen et al. (2008) extend Stochastic Logic Programs (Muggleton, 2003) to incorporate abduction. Sato (1995) has also developed a probabilistic logic called PRISM that performs abduction. However, none of these approaches have been applied to plan recognition.

3.7 Summary

We introduced a new approach to plan recognition using Bayesian Logic Programs (BLPs) in this chapter. We extended BLPs for plan abductive plan recognition by employing logical abduction to construct the Bayesian networks as opposed to the standard logical deduction used in BLPs. We call the resulting model

Bayesian Abductive Logic Programs (BALPs). We also demonstrated that the parameters of the BALP model can be learned automatically using the EM algorithm. Empirical evaluations on three benchmark data sets from different application domains demonstrated that BALPs generally outperform the state-of-the-art for plan recognition. We believe that the superior performance achieved by BALPs is due to the combination of logical abduction, joint probabilistic inference, and incorporation of domain knowledge. Overall, we found that BALPs were very effective for plan recognition.

Chapter 4

Machine Reading using Bayesian Logic Programs

4.1 Introduction

In the previous chapter, we demonstrated the efficacy of BLPs on plan recognition, which is an abductive reasoning task. In this chapter, we develop an approach to machine reading using BLPs and demonstrate its efficacy on a deductive reasoning task.

The task of machine reading involves automatic extraction of knowledge or information from natural language text. The internet has grown exponentially in the last few years, resulting in the accumulation of large amounts of on-line text. One way to search for information on a particular topic on the web is by using a search engine like Google[1] that returns relevant documents to the user. However, the user still has to read through all the documents to get the specific information he/she is looking for. Instead, it would be convenient to have a machine reading system that accepts a query/question from the user and returns the specific answer that the user is looking for. Such a system could be useful for several types of professionals such as doctors who could use it to stay on top of the latest developments in medicine. People who work for security agencies could use it to keep track of various terror-

[1] `www.google.com`

istic events that happen around the world. Due to these reasons, the last few years have attracted a lot of interest in machine reading.

Much of the information conveyed in natural language text must be inferred from what is explicitly stated since easily inferable facts are rarely mentioned. This was first discussed in detail by Grice (1975), who postulated the maxims of quantity, quality, manner, and relation that characterize natural-language communication. The maxim of quantity refers to the concise nature of natural language that leaves much implicit information unstated. Human readers typically "read between the lines" and infer information that is implicit from explicitly stated facts using common sense knowledge.

Automated information extraction (IE) systems (Cowie & Lehnert, 1996; Sarawagi, 2008), which are a type of machine reading system are trained to extract information that is explicitly stated in the text. Further, these systems do not have access to common sense knowledge, and hence are not capable of performing deeper inference. As a result, they are limited in their ability to extract implicit facts. However, answering many queries can require inferring such implicit stated facts. Consider the text "Barack Obama is the president of the United States of America." Given the query "Barack Obama is a citizen of what country?", standard IE systems cannot identify the answer since citizenship is not explicitly stated in the text. However, a human reader possesses the common sense knowledge that the president of a country is almost always a citizen of that country, and easily infers the correct answer.

In this chapter, we consider the problem of inferring implicit facts from nat-

ural language text. The standard approach to inferring implicit information involves using common sense knowledge in the form of logical rules to deduce additional information from the extracted facts. Since manually developing such a knowledge base is difficult and arduous, an effective alternative is to *learn* automatically such rules by mining a substantial database of facts that an IE system has already automatically extracted from a large corpus of text (Nahm & Mooney, 2000). Due to the concise and incomplete nature of natural language text, facts that are easily inferred from explicitly stated facts are rarely mentioned in the text. Hence, the facts extracted by an IE system are always quite noisy and incomplete. As a result, the main challenge in this task involves learning first-order rules from a few instances of inferable facts seen in the training data.

Some of the existing approaches modify Inductive Logic Programming (ILP) (Lavrač & Džeroski, 1994) based rule learners to learn probabilistic first-order rules and subsequently perform purely logical deduction to infer new facts (Carlson, Betteridge, Kisiel, Settles, Jr., & Mitchell, 2010; Doppa, NasrEsfahani, Sorower, Dietterich, Fern, & Tadepalli, 2010). These approaches do not use a well-founded probabilistic graphical model to compute coherent probabilities for inferred facts. Alternate approaches involve using the MLN framework (Schoenmackers, Etzioni, Weld, & Davis, 2010; Sorower, Dietterich, Doppa, Walker, Tadepalli, & Fern, 2011) for both learning first-order rules and probabilistic inference of additional facts. While MLNs can handle noisy and incomplete IE extractions, they seldom scale to large datasets since the "brute force" grounding process in MLNs include all possible type-consistent groundings of the rules in the corresponding Markov net, which

could result in an intractably large graphical model for larger datasets.

In order to alleviate limitations with the existing approaches, we propose a novel approach that uses BLPs to infer implicit facts. Our approach uses LIME (McCreath & Sharma, 1998), an ILP rule learner that is capable of handling noise to learn first-order rules and then use the BLP framework to infer additional facts. Unlike purely logical deduction, BLPs employ a well-founded probabilistic graphical model such as Bayesian networks that compute coherent probabilities for new facts. Unlike MLNs, BLPs employ focused grounding by including only those literals that are used to prove the query, thereby making them scalable to large corpora of natural language text. Hence, we hypothesize that BLPs are better suited for the task of inferring implicit facts from natural language text.

The main contribution of this chapter is the effective application of BLPs for inferring implicit information from natural language text. We demonstrate that it is possible to learn the structure and the parameters of BLPs automatically using only noisy and incomplete extractions from natural language text, which we then use to infer additional facts from text.

We have implemented this approach by using an off-the-shelf IE system and developing novel adaptations of existing learning methods to construct efficiently fast and effective BLPs for reading between the lines. We present an experimental evaluation of our resulting system on a realistic test corpus from DARPA's Machine Reading project, and demonstrate improved performance compared to a purely logical approach based on ILP, and an alternative SRL approach based on MLNs.

The rest of this chapter is organized as follows. Section 4.2 describes our BLP-based approach to learning to infer implicit facts. Section 4.3 describes our experimental methodology and discusses the results of our evaluation. Section 4.5 discusses related work and highlights key differences between our approach and existing work.

4.2 Learning BLPs to Infer Implicit Facts

We describe our approach to inferring implicit facts from natural language text using BLPs. Figure 4.1 shows the overall architecture of our system. Given a set of training documents in natural language text, the first step involves extracting explicitly mentioned facts using an IE system. We use IBM's SIRE (Florian, Hassan, Ittycheriah, Jing, Kambhatla, Luo, Nicolov, & Roukos, 2004) to extract IE extractions from natural language text. Next, we learn common sense knowledge in the form of first-order logical rules using an ILP based rule learner called LIME (McCreath & Sharma, 1998) as described in Section 4.2.1. Given the learned first order rules, we specify parameters using the techniques described in Section 4.2.2. Finally, when IE extractions from a new document are given, the learned first-order rules along with the respective parameters are used to deduce additional facts using the BLP inference engine as described in Section 4.2.3. Note that the learned first-order rules together with their parameters completely specify the BLP for inferring implicit facts.

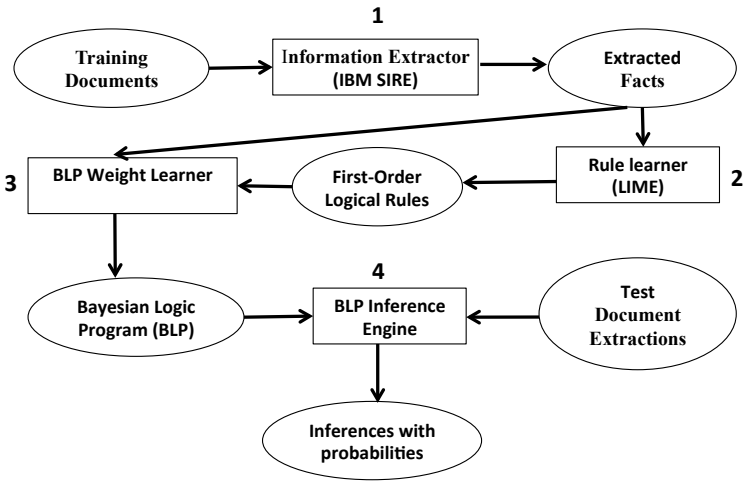


Figure 4.1: System architecture for inferring implicit facts using BLPs

4.2.1 Learning Rules from Extracted Data

We now discuss our approach to learning common sense knowledge in the form of first-order definite clauses from text. We first extract facts that are explicitly stated in the text using SIRE (Florian et al., 2004), an IE system developed by IBM. We then learn first-order rules from these extracted facts using LIME (McCreath & Sharma, 1998), an ILP system designed for noisy training data. As described in Chapter 2, LIME is capable of learning from only positive and both positive and negative examples. Further, LIME does not require the same number of positive and negative examples for learning first-order rules. Typically, in the machine reading task, we have access to only positive instances. Negative instances are artificially generated (see below) using the closed world assumption. Further, the ratio of positive to negative instances is typically skewed. For these reasons, we used LIME to learn first-order rules for this task.

We first identify a set of target relations we want to infer. Typically, an ILP system takes a set of positive and negative instances for a target relation, along with a background knowledge base (in our case, other facts extracted from the same document) from which the positive instances are potentially inferable. In our task, we only have direct access to positive instances of target relations, i.e. the relevant facts extracted from the text. So we artificially generate negative instances using the *closed world assumption*, which states that any instance of a relation that is not extracted can be considered a negative instance. We note that the closed world assumption does not necessarily hold for this task since several implicit facts could be labeled as negative instances. However, it typically generates a useful (if noisy) set of negative instances. For each relation, we generate all possible type-consistent instances using all constants in the domain. All instances that are not extracted facts (i.e. positive instances) are labeled as negative. The total number of such closed world negatives can be intractably large, so we randomly sample a fixed-sized subset. The ratio of 1:20 for positive to negative instances worked well in our approach.

Since LIME can learn rules using only positive instances, or using both positive and negative instances, we learn rules using both settings. LIME learns fewer rules that are very general when given only positive instances, while it learns more specific rules when given both positive and negative instances. As a result, we include all unique rules learned from both settings in the final set, since the goal of this step is to learn a large set of potentially useful rules whose relative strengths will be determined in the next step of parameter learning. Other approaches could

also be used to learn candidate rules. We initially tried using the popular ALEPH ILP system (Srinivasan, 2001), but it did not produce useful rules, probably due to the high level of noise in our training data.

4.2.2 Learning BLP Parameters

As described in Chapter 2, we use a deterministic logical-and model to encode the CPT entries associated with Bayesian clauses, and use *noisy-or* to combine evidence coming from multiple ground rules that have the same head (Pearl, 1988). The noisy-or model requires just a single parameter for each rule, which can be learned from training data.

We learn the noisy-or parameters using the EM algorithm adapted for BLPs by Kersting and De Raedt (2008) (see Chapter 2). In our task, the supervised training data consists of facts that are extracted from the natural language text. However, we usually do not have evidence for inferred facts as well as noisy-or nodes. As a result, there are a number of variables in the ground networks which are always hidden, and hence EM is appropriate for learning the requisite parameters from the partially observed training data.

4.2.3 Inference of Additional Facts using BLPs

Inference in the BLP framework involves backward chaining (Russell & Norvig, 2003) from a specified query (SLD resolution) to obtain all possible deductive proofs for the query. In the context of the current task, each target relation becomes a query on which we backchain. We then construct a ground Bayesian

network using the resulting deductive proofs for all target relations and learned parameters using the standard approach described in Chapter 2. Finally, we perform standard probabilistic inference to estimate the marginal probability of each inferred fact. Since exact inference was intractable, we use Sample Search (Gogate & Dechter, 2007) to perform probabilistic inference.

4.2.4 Illustrative example

We now illustrate our approach to infer implicit facts with a concrete example. Figure 4.2a shows an example test document and Figure 4.2b shows the corresponding extractions, extracted using an IE system. Figure 4.2c shows common sense knowledge learned in the form of first-order rules. Given the extractions and the first-order rules learned, BLP inference engine performs deductive reasoning and constructs the ground Bayesian network shown in Figure 4.2d. The ground Bayesian network shows that the BLP inference engine has inferred *hasCitizenship(barack obama, usa)* using two different rules. After learning the noisy-or parameters, probabilistic inference is performed to estimate the marginal probability of *hasCitizenship(barack obama, usa)*. In Figure 4.2d, nodes with thick borders represent evidence nodes, nodes with dotted borders represent intermediate logical-and and noisy-or nodes, and nodes with thin borders represent inferences for which marginal probabilities are estimated.

(a) *Example test document*
"Barack Obama is the president of USA."

(b) *IE extractions*
nationState(usa)
person(barack obama)
isLedBy(usa,barack obama)
employs(usa,barack obama)

(c) *First order rules constructed*
isLedBy(X,Y) ∧ person(Y) ∧ nationState(X) → hasCitizenship(Y,X)
employs(X,Y) ∧ person(Y) ∧ nationState(X) → hasCitizenship(Y,X)

(d) *Ground Bayesian network constructed from BLP inference*

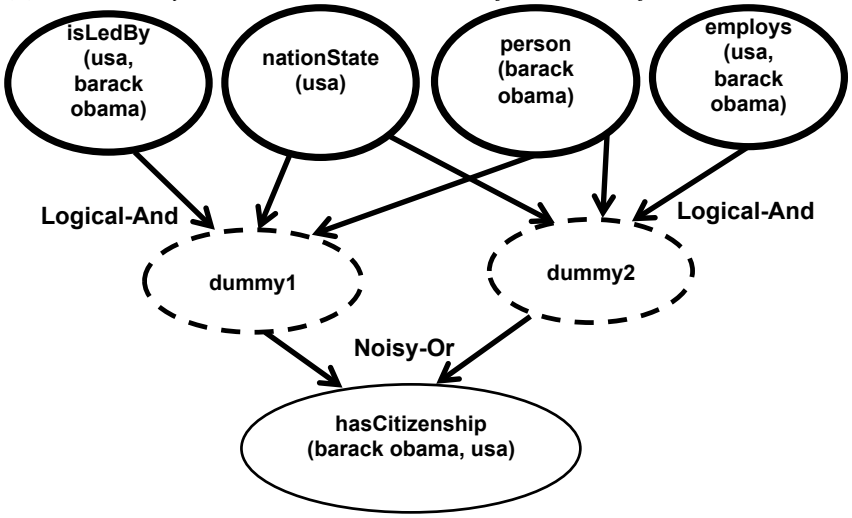


Figure 4.2: Example describing the inference of new facts using BLPs.

4.3 Experimental Evaluation

4.3.1 Data

For evaluation, we used DARPA's machine-reading intelligence-community (IC) data set, which consists of news articles on terrorist events around the world. There are $10,000$ documents each containing an average of 89.5 facts extracted by SIRE (Florian et al., 2004). SIRE assigns each extracted fact a confidence score and we used only those with a score of 0.5 or higher for learning and inference. An average of 86.8 extractions per document meet this threshold.

DARPA also provides an ontology describing the entities and relations in the IC domain. It consists of 57 entity types and 79 relations. The entity types include *Agent*, *PhysicalThing*, *Event*, *TimeLocation*, *Gender*, and *Group*, each with several subtypes. The type hierarchy is a DAG rather than a tree, and several types have multiple super-classes. For instance, a *GeopoliticalEntity* can be a *HumanAgent* as well as a *Location*. This can cause some problems for systems that rely on a strict typing system, such as MLNs (as implemented in Alchemy) which rely on types to limit the space of ground literals that are considered. Some sample relations are *attendedSchool*, *approximateNumberOfMembers*, *mediatingAgent*, *employs*, *hasMember*, *hasMemberHumanAgent*, and *hasBirthPlace*.

4.3.2 Methodology

We evaluated our approach using 10-fold cross validation. We learned first-order rules for the 13 target relations shown in Table 4.3 from the facts extracted from the training documents (Section 4.2.1). These relations were selected since

governmentOrganization(A) ∧ employs(A,B) → hasMember(A,B) *If a government organization A employs person B, then B is a member of A*
eventLocation(A,B) ∧ bombing(A) → thingPhysicallyDamaged(A,B) *If a bombing event A took place in location B, then B is physically damaged*
isLedBy(A,B) → hasMemberPerson(A,B) *If a group A is led by person B, then B is a member of A*
nationState(B) ∧ eventLocationGPE(A,B) → eventLocation(A,B) *If an event A occurs in a geopolitical entity B, then the event location for that event is B*
mediatingAgent(A,B) ∧ humanAgentKillingAPerson(A) → killingHumanAgent(A,B) *If A is an event in which a human agent is killing a person and the mediating agent of A is an agent B, then B is the human agent that is killing in event A*

Table 4.1: A sample set of rules learned using LIME

they have an appreciable amount of data. Since LIME does not scale well to large data sets, we could train it on at most about $2,500$ documents. Consequently, we split the $9,000$ training documents into four disjoint subsets and learned first-order rules from each subset. The final knowledge base included all unique rules learned from any subset. LIME learned several rules that had only entity types in their bodies. Such rules make many incorrect inferences; hence we eliminated them. We also eliminated rules violating type constraints. We learned an average of 48 rules per fold. Table 4.1 shows some sample learned rules.

We then learned parameters as described in Section 4.2.2. We initially set all noisy-or parameters to 0.9 based on the intuition that if exactly one rule for a consequent was satisfied, it could be inferred with a probability of 0.9.

For each test document, we performed BLP inference as described in Section 4.2.3. We ranked all inferences by their marginal probability, and evaluated the results by either choosing the top n inferences or accepting inferences whose

marginal probability was equal to or exceeded a specified threshold. We evaluated two BLPs with different parameter settings: *BLP-Learned-Weights* used noisy-or parameters learned using EM, *BLP-Manual-Weights* used fixed noisy-or weights of 0.9.

4.3.3 Annotation of ground truth for evaluation

The IC data set lacks ground truth information, i.e. information about all facts that can be inferred from a given document. As a result, automatic evaluation of inferred facts is not possible. It is possible to manually evaluate inferred facts to estimate *precision*, which measures the fraction of inferred facts that are correct. However, this methodology does not allow for estimation of *recall*, which measures the fraction of all facts that can be inferred from the document correctly. For automatic evaluation of precision and recall, we explored the possibility of annotating a subset of documents with all possible facts that can be inferred from the respective documents.

We first randomly sampled two documents from the test set in each fold. For each target relation, we generated all possible type-consistent instances using all constants extracted by the extractor from the document. We then manually evaluated each instance based on the natural language text in the document and retained those instances that were found to be true as ground truth. Typically, the list of all possible relation instances generated per document ranged between 5000 – 9000. For some larger documents with several constants, it was close to $20,000$. The ground truth after discarding incorrect instances typically reduced to a set of 100

– 200 facts for each document. We took approximately 6-7 hours to annotate one document and we initially tried to annotate 20 documents in total.

We ran into several issues during manual annotation, which prevented us from creating an accurately annotated data set for automatic evaluation. We found that there were several relation instances that were explicitly stated in the document, but were not extracted by the extractor. We could have excluded these instances in the ground truth set since our task involved evaluating facts that could be inferred from the document. However, we realized that these relations that were not extracted by the extractor could be inferred based on the other facts that were extracted. As a result, it was difficult to distinguish between facts that could be strictly inferred from those that were explicitly stated. Hence, we included all instances that were found to be *true* for a given document. This process could have possibly included those facts that were strictly explicitly stated in the document, but could not be inferred based on the remaining facts. Further, we observed that even though some inferences could be made based on the explicitly stated facts, the same inferences could not be made based on the extracted facts since the extractor had not extracted some of the explicitly stated facts. Here again, we could not have included those inferences that were not supported by the extracted facts in the ground truth. However, this would have resulted in different ground truth for different sets of extracted facts. Finally, the extractor had made several mistakes during the extraction process. For instance, it extracted "car", which was used as a vehicle in the document, as an instance of "weapon". It was not clear how to incorporate the extractor's mistakes into our annotation. When evaluating instances that involved

"car", we treated "car" as an instance of vehicle and not that of weapon.

Relations such as *attendedSchool* and *hasBirthPlace* had very few instances in the ground truth. On the other hand, many instances of relations such as *employs* and *hasMember* were present in the ground truth. This suggests that relations such as *attendedSchool* and *hasBirthPlace* are not easily inferable in nature. Unless they are explicitly mentioned in the document, it is highly unlikely that such relations will be inferred based on other facts unlike instances of relations such as *employs* and *hasMember*. For the examples annotated with ground truth, the overall recall was extremely poor, in the range between 1-5%. The low recall score was partially due to the difficulties described above with respect to annotation of ground truth. Another possibility for the low recall could arise from the lack of a more expressive ontology that is capable of capturing a wide variety of relations. For instance, there is a very subtle difference between the relations *thingPhysicallyDamaged* and *thingPhysicallyDestroyed*. An instance of *thingPhysicallyDestroyed* could also be an instance of *thingPhysicallyDamaged*. Relations describing family relationships such as *hasFather*, *hasSon*, *hasSpouse*, etc. cannot be inferred due to a lack of sufficient information in the IC ontology. Due to these issues, it was difficult to measure the *correct* recall of our system. As a result, we decided not to pursue this approach to evaluation. Instead, we manually evaluated inferences to calculate precision as described below. Since the number of inferences made for each document ranged roughly from 20-100, we could scale our manual evaluation for the calculation of precision to a larger set of documents.

4.3.4 Evaluation Metrics

As described above, the lack of ground truth annotation for inferred facts prevents an automated evaluation, so we resorted to a manual evaluation. We randomly sampled 40 documents (4 from each test fold), judged the accuracy of the inferences for those documents, and computed *precision*, the fraction of inferences that were deemed correct. For probabilistic methods such as BLPs and MLNs that provide certainties for their inferences, we also computed *precision at top n*, which measures the precision of the n inferences with the highest marginal probability across the 40 test documents. Our evaluation is similar to that used in previous related work (Carlson et al., 2010; Schoenmackers et al., 2010).

SIRE frequently makes incorrect extractions, and therefore inferences made from these extractions are also inaccurate. To account for the mistakes made by the extractor, we report two different precision scores. The "unadjusted" (UA) score, does not correct for errors made by the extractor. The "adjusted" (AD) score does not count mistakes due to extraction errors. That is, if an inference is incorrect because it was based on incorrect extracted facts, we remove it from the set of inferences and calculate precision for the remaining inferences.

4.3.5 Baselines

Since none of the existing approaches have been evaluated on the IC data, we cannot directly compare our performance to theirs. Therefore, we compared to the following methods:

- *Logical Deduction*: This method forward chains on the extracted facts using the first-order rules learned by LIME to infer additional facts. This approach is unable to provide any confidence or probability for its conclusions.

- *Markov Logic Networks (MLNs)*: We use the rules learned by LIME to define the structure of an MLN. In the first setting, which we call *MLN-Learned-Weights*, we learn the MLN's parameters using the generative weight learning algorithm (Domingos & Lowd, 2009), which we modified to process training examples in an online manner. In online generative learning, gradients are calculated and weights are estimated after processing each example and the learned weights are used as the starting weights for the next example. The pseudo-likelihood of one round is obtained by multiplying the pseudo-likelihood of all examples. In our approach, the initial weights of clauses are set to 10. Convergence to optimal weights was reached after 131 iterations, on average. In the second setting, which we call *MLN-Manual-Weights*, we assign a weight of 10 to all rules since it worked the best in our experiments. We used maximum likelihood prior for all predicates.

  MLN-Manual-Weights is similar to BLP-Manual-Weights in that all rules are given the same weight. We then use the learned rules and parameters to infer probabilistically additional facts using the MC-SAT algorithm implemented in Alchemy,[2] an open-source MLN package.

[2] `http://alchemy.cs.washington.edu/`

	UA	AD
Precision	29.73 (443/1490)	35.24 (443/1257)

Table 4.2: Precision for logical deduction. "UA" and "AD" refer to the unadjusted and adjusted scores respectively

4.4 Results and Discussion

4.4.1 Comparison to Baselines

Table 4.2 gives the unadjusted (UA) and adjusted (AD) precision for logical deduction. Out of $1,490$ inferences for the 40 evaluation documents, 443 were judged correct, giving an unadjusted precision of 29.7%. Out of these $1,490$ inferences, 233 were determined to be incorrect due to extraction errors, improving the adjusted precision to a modest 35.2%.

MLNs made about $127,000$ inferences for the 40 evaluation documents. Since it is not feasible to evaluate manually *all* the inferences made by the MLN, we calculated precision using only the top $1,000$ inferences. Since BLP uses logical deduction to construct the ground Bayesian networks, the total number of inferences made by the BLP approach is same as that made by purely logical deduction (see Table 4.2). Figure 4.3 shows both unadjusted and adjusted precision at top-n for various values of n for different BLP and MLN models. For both BLPs and MLNs, simple manual weights result in superior performance than the learned weights. Despite the fairly large size of the overall training sets ($9,000$ documents), the amount of data for each target relation is apparently still not sufficient to learn particularly accurate weights for both BLPs and MLNs. However, for BLPs, learned weights do show a substantial improvement initially (i.e. top 25–50 inferences), with an aver-

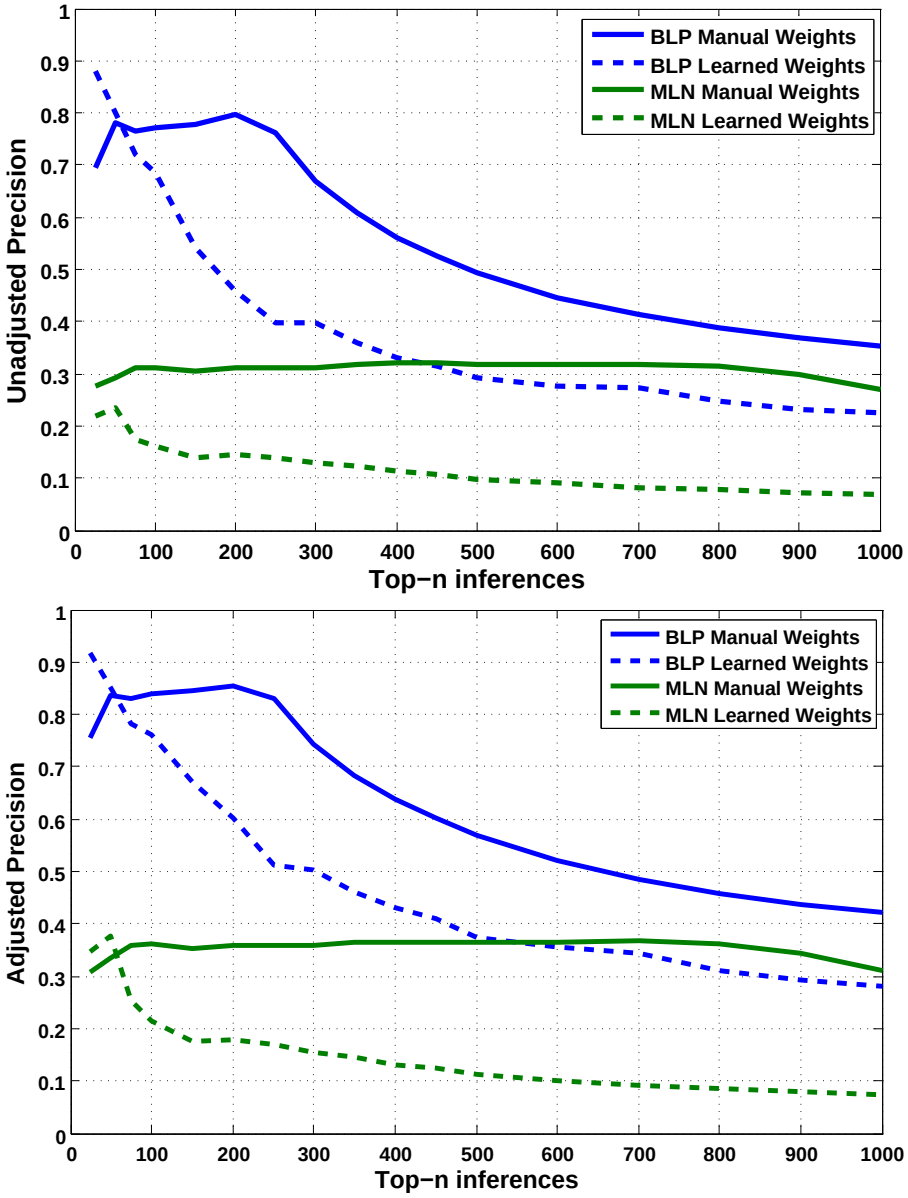


Figure 4.3: Unadjusted and adjusted precision at top-n for different BLP and MLN models for various values of n

age of 1 inference per document at 91% adjusted precision as opposed to an average of 5 inferences per document at 85% adjusted precision for BLP-Manual-Weights. For MLNs, learned weights show a small improvement initially only with respect to adjusted precision. Between BLPs and MLNs, BLPs perform substantially better than MLNs at most points in the curve. However, MLN-Manual-Weights improve marginally over BLP-Learned-Weights at later points (top 600 and above) on the curve, where the precision is generally very low. Here, the superior performance of BLPs over MLNs could be possibly due to the focused grounding used in the BLP framework.

For BLPs, as n increases towards including all of the logically sanctioned inferences, as expected, the precision converges to the results for logical deduction. However, as n decreases, both adjusted and unadjusted precision increase fairly steadily. This demonstrates that probabilistic BLP inference provides a clear improvement over logical deduction, allowing the system to accurately select the best inferences that are most likely to be correct. Unlike the two BLP models, MLN-Manual-Weights has more or less the same performance at most points on the curve, and it is slightly better than that of purely-logical deduction. MLN-Learned-Weights is worse than purely-logical deduction at most points on the curve.

4.4.2 Results for Individual Target Relations

Table 4.3 shows both unadjusted and adjusted precision for each relation for instances inferred using logical deduction, BLP-Manual-Weights, and BLP-Learned-Weights with a confidence threshold of 0.95. The probabilities estimated

for inferences by MLNs are not directly comparable to those estimated by BLPs. As a result, we do not include results for MLNs here. For this evaluation, using a confidence threshold based cutoff is more appropriate than using top-n inferences made by the BLP models since the estimated probabilities can be directly compared across target relations.

For logical deduction, precision is high for a few relations such as *employs*, *hasMember*, and *hasMemberHumanAgent*, indicating that the rules learned for these relations are more accurate than the ones learned for the remaining relations. Unlike relations such as *hasMember* that are easily inferred from relations such as *employs* and *isLedBy*, certain relations such as *hasBirthPlace* are not easily inferable using the information in the ontology. As a result, it might not be possible to learn accurate rules for such target relations. Other reasons include the lack of a sufficiently large number of target-relation instances during training and lack of strictly defined types in the IC ontology.

Both BLP-Manual-Weights and BLP-Learned-Weights also have high precision for several relations (*eventLocation*, *hasMemberHumanAgent*, *thingPhysicallyDamaged*). However, the actual number of inferences can be fairly low. For instance, 103 instances of *hasMemberHumanAgent* are inferred by logical deduction (i.e. 0 confidence threshold), but only 2 of them are inferred by BLP-Learned-Weights at 0.95 confidence threshold, indicating that the parameters learned for the corresponding rules are not very high. For several relations such as *hasMember*, *hasMemberPerson*, and *employs*, no instances were inferred by BLP-Learned-Weights at 0.95 confidence threshold. On the other hand, BLP-Manual-Weights has

inferred 26 instances of *hasMemberHumanAgent*.

Lack of sufficient training instances (extracted facts) is possibly one of the reasons for learning low weights for rules for target relations such as *hasMember* and *hasMemberPerson*. Another possible reason is that relations such as *hasMember* are more likely to be inferred due to which they are seldom stated explicitly in the text. As a result, they are rarely seen in the training data. Increasing the number of training instances might solve the problem to some extent. However, the proportion of instances for such relations that are more likely to be inferred are still going to be smaller when compared to that for relations that are explicitly stated in the text. As a result, EM might still end up learning low weights for rules for target relations that are more likely to be inferred. Providing additional supervision to EM in the form of ground truth information, i.e. relations that can be inferred from explicitly stated facts might alleviate this problem and we consider this as a topic for future work.

4.4.3 Discussion

We now discuss the potential reasons for BLP's superior performance compared to other approaches. Probabilistic reasoning used in BLPs allows for a principled way of determining the most confident inferences, thereby allowing for improved precision over purely logical deduction. The primary difference between BLPs and MLNs lies in the approaches used to construct the ground network. In BLPs, only propositions that can be logically deduced from the extracted evidence are included in the ground network. On the other hand, MLNs include all possible

Relation	UA/AD	Logical Deduction	BLP-Manual-Weights-.95	BLP-Learned-Weights-.95	No. train inst.
employs	UA	65.78 (25/38)	92.85 (13/14)	nil (0/0)	18,440
	AD	69.44 (25/36)	92.85 (13/14)	nil (0/0)	
eventLocation	UA	15.00 (18/120)	100.00 (1/1)	100 (1/1)	6902
	AD	18.75 (18/96)	100.00 (1/1)	100 (1/1)	
hasMember	UA	87.96 (95/108)	92.20 (71/77)	nil (0/0)	1462
	AD	95.95 (95/99)	97.26 (71/73)	nil (0/0)	
hasMemberPerson	UA	40.00 (42/105)	93.33 (14/15)	nil (0/0)	705
	AD	43.75 (42/96)	100 (14/14)	nil (0/0)	
isLedBy	UA	12.30 (8/65)	nil (0/0)	nil (0/0)	8402
	AD	12.30 (8/65)	nil (0/0)	nil (0/0)	
mediatingAgent	UA	13.15 (15/114)	nil (0/0)	nil (0/0)	92,998
	AD	19.73 (15/76)	nil (0/0)	nil (0/0)	
thingPhysicallyDamaged	UA	21.60 (62/287)	90.32 (28/31)	90.32 (28/31)	24,662
	AD	25.72 (62/241)	90.32 (28/31)	90.32 (28/31)	
hasMemberHumanAgent	UA	86.72 (98/113)	89.65 (26/29)	66.66 (2/3)	3619
	AD	95.14 (98/103)	100.00 (26/26)	100.00 (2/2)	
killingHumanAgent	UA	11.81 (43/364)	25.00 (2/8)	40.00 (2/5)	3341
	AD	15.35 (43/280)	33.33 (2/6)	66.67 (2/3)	
hasBirthPlace	UA	0.00 (0/89)	nil (0/0)	nil (0/0)	89
	AD	0.00 (0/88)	nil (0/0)	nil (0/0)	
thingPhysicallyDestroyed	UA	nil (0/0)	nil (0/0)	nil (0/0)	800
	AD	nil (0/0)	nil (0/0)	nil (0/0)	
hasCitizenship	UA	42.52 (37/87)	50.72 (35/69)	nil (0/0)	222
	AD	48.05 (37/77)	58.33 (35/60)	nil (0/0)	
attendedSchool	UA	nil (0/0)	nil (0/0)	nil (0/0)	2
	AD	nil (0/0)	nil (0/0)	nil (0/0)	

Table 4.3: Unadjusted (UA) and adjusted (AD) precision for individual relations

type-consistent groundings of all rules in the network, introducing many ground literals which cannot be logically deduced from the evidence. This generally results in several incorrect inferences, thereby yielding poor performance.

Even though learned weights in BLPs do not result in a superior performance, learned weights in MLNs are substantially worse. Lack of sufficient training data is one of the reasons for learning less accurate weights by the MLN weight learner. However, a more important issue is due to the use of the closed world assumption during learning, which we believe is adversely impacting the weights learned. As mentioned earlier, for the task considered in this chapter, if a fact is not explicitly stated in text, and hence not extracted by the extractor, it does not necessarily imply that it is not true. Since existing weight learning approaches for MLNs do not deal with missing data and open world assumption, developing such approaches is a topic for future work.

Apart from developing novel approaches for weight learning, additional engineering could potentially improve the performance of MLNs on the IC data set. Due to MLN's grounding process, several spurious facts like *employs(a,a)* were inferred. These inferences can be prevented by including additional clauses in the MLN that impose integrity constraints that prevent such nonsensical propositions. Further, techniques proposed by Sorower et al. (2011) can be incorporated to explicitly handle missing information in text. Lack of strict typing on the arguments of relations in the IC ontology has also resulted in inferior performance of the MLNs. To overcome this, relations that do not have strictly defined types could be specialized. Finally, we could use the deductive proofs constructed by BLPs to constrain

the ground Markov network, similar to the model-construction approach adopted in MLN-HCAM by Singla and Mooney (2011) (see Chapter 3).

However, in contrast to MLNs, BLPs that use first-order rules that are learned by an off-the-shelf ILP system and given simple intuitive hand-coded weights, are able to provide fairly high-precision inferences that augment the output of an IE system and allow it to effectively infer implicit facts in natural language text.

4.5 Related Work

Several previous projects (Nahm & Mooney, 2000; Carlson et al., 2010; Schoenmackers et al., 2010; Doppa et al., 2010; Sorower et al., 2011) have mined inference rules from data automatically extracted from text by an IE system. Similar to our approach, these systems use the learned rules to infer additional information from facts directly extracted from a document. Nahm and Mooney (2000) learn *propositional rules* using C4.5 (Quinlan, 1993) from data extracted from computer-related job-postings, and therefore cannot learn multi-relational rules with quantified variables. Other systems (Carlson et al., 2010; Schoenmackers et al., 2010; Doppa et al., 2010; Sorower et al., 2011) learn *first-order rules* (i.e. Horn clauses in first-order logic).

Carlson et al. (2010) modify an ILP system similar to FOIL (Quinlan, 1990) to learn rules with probabilistic conclusions. They use purely logical deduction (forward-chaining) to infer additional facts. Unlike BLPs, this approach does not use a well-founded probabilistic graphical model to compute coherent probabilities for inferred facts. Further, Carlson et al. (2010) used a human judge to manu-

ally evaluate the quality of the learned rules before using them to infer additional facts. Our approach, on the other hand, is completely automated and learns fully parameterized rules in a well-defined probabilistic logic.

Schoenmackers et al. (2010) develop a system called SHERLOCK that uses statistical relevance to learn first-order rules. Unlike our system and others (Carlson et al., 2010; Doppa et al., 2010; Sorower et al., 2011) that use a pre-defined ontology, they automatically identify a set of entity types and relations using "open IE." They use HOLMES (Schoenmackers, Etzioni, & Weld, 2008), an inference engine based on MLNs to infer additional facts. However, as mentioned earlier, MLNs include all possible type-consistent groundings of the rules in the corresponding Markov net, which, for larger datasets, can result in an intractably large graphical model. To overcome this problem, HOLMES uses a specialized model construction process to control the grounding process. Unlike MLNs, BLPs naturally employ a more focused approach to grounding by including only those literals that are used to deduce the query.

Doppa et al. (2010) use FARMER (Nijssen & Kok, 2003), an existing ILP system, to learn first-order rules. They propose several approaches to score the rules, which are used to infer additional facts using purely logical deduction. Sorower et al. (2011) propose a probabilistic approach to modeling implicit information as missing facts and use MLNs to infer these missing facts. They learn first-order rules for the MLN by performing exhaustive search, which might be computationally intensive for large domains. As mentioned earlier, inference using both these approaches, logical deduction and MLNs, have certain limitations, which BLPs

help overcome. Unlike our approach, both Doppa et al. (2010) and Sorower et al. (2011) have evaluated their approaches using a couple of target relations.

DIRT (Lin & Pantel, 2001) and RESOLVER (Yates & Etzioni, 2007) learn inference rules, also called entailment rules that capture synonymous relations and entities from text. Berant et al. (2011) propose an approach that uses transitivity constraints for learning entailment rules for typed predicates. Unlike the systems described above, these systems do not learn complex first-order rules that capture common sense knowledge. Further, most of these systems do not use extractions from an IE system to learn entailment rules, thereby making them less related to our approach.

4.6 Summary

In this chapter, we have introduced a novel approach using BLPs to learning to infer implicit information from facts extracted from natural language text. We have demonstrated that it can learn useful first-order rules from a large database of noisy and incomplete IE extractions. Our experimental evaluation on the IC data set demonstrates the advantage of BLPs over logical deduction and an approach based on MLNs.

Chapter 5

Online Inference-Rule Learning from Natural Language Extractions

5.1 Introduction

In the previous chapter, we demonstrated that BLPs are a good formalism for inferring implicit facts from natural language text. We used LIME, an ILP based rule learner to learn common sense knowledge in the form of first order rules. Most existing rule learners (Quinlan, 1990; McCreath & Sharma, 1998; Srinivasan, 2001; Kersting & De Raedt, 2008; Dinh, Exbrayat, & Vrain, 2011) assume that the training data is largely accurate. Since much of the information conveyed in text must be inferred from what is explicitly stated, entities and relations extracted using an information extraction (IE) system are incomplete and noisy. As a result, most existing rule learners are not adept at learning useful rules from natural language extractions. Further, most of them do not scale to large corpora.

The limitations described above for ILP based rule learners are present in LIME as well. The rules learned by LIME using only positive instances are very few and very general. Since we do not have access to negative instances for any given target relation, we artificially generate negative instances using the closed world assumption (see Chapter 4). However, the closed world assumption does not

always hold for natural language text since the instances that are not extracted by the extractor are not necessarily false; they might not have been explicitly stated in the text. As a result, LIME does not always learn rules that handle the concise, incomplete nature of natural-language text. Further, similar to existing rule learners, LIME does not scale to large corpora. As described in the previous chapter (Chapter 4), we split the training documents into smaller subsets and then learned first-order rules separately on each subset using LIME.

In this chapter, we develop a novel approach to learning common sense knowledge in the form of first-order rules from incomplete natural language extractions for the machine reading task. These rules are then used to infer implicit facts from natural language text (Carlson et al., 2010; Schoenmackers et al., 2010; Doppa et al., 2010; Sorower et al., 2011; Raghavan et al., 2012). The proposed rule learner learns probabilistic first-order definite clauses from incomplete IE extractions in which the body of the clause typically consists of relations that are frequently explicitly stated, while the head is a relation that is more typically inferred. We use the frequency of occurrence of extracted relations as a *heuristic* for distinguishing those that are typically explicitly stated from the ones that are usually inferred. In order to allow scaling to large corpora, we develop an efficient online rule learner. Unlike existing rule learners that require both positive and negative instances to be specified, our rule learner learns rules from only positive instances.

For each example in training, we construct a *directed* graph of relation extractions and add directed edges between nodes that share one or more constants. Then we traverse the graph to learn first-order rules. Our approach is closest to the

work by Dinh et al. (2011) in which they construct an *undirected* graph of first-order predicates and add edges between nodes whose predicates share arguments of the same type. In our approach, the directionality of the edges helps discover rules for those relations that can be inferred from others that are explicitly stated in the text. Further, by constructing a graph of ground relation literals instead of relation predicates, our learner can be used in domains that do not have a strictly tree-structured ontology. Typically, relations that accept arguments or constants belonging to multiple types are found in ontologies which have the structure of a DAG (directed acyclic graph) rather than a tree. Accommodating such an ontology is critical to handling the machine-reading corpus used in our experiments. The approach by Dinh et al. is not directly applicable to such domains since it relies on a unique type for each predicate argument.

After learning first-order rules, additional facts are inferred by performing deductive reasoning using the learned rules as common sense knowledge. As mentioned in the previous chapter, approaches to inference use either purely logical deduction, which fails to account for the uncertainty inherent in such rules, or a well founded probabilistic logic such as MLNs or BLPs. As demonstrated in the previous chapter, BLPs have a superior performance wrt accuracy over both MLNs and purely logical deduction. Hence, we use BLPs to infer implicit facts from natural language text in this chapter as well.

Probabilistic inference using the BLP framework requires learning parameters (CPT entries) for the first-order rules. Since those relations that are easily inferred from explicitly stated facts are seldom seen in the training data, learning

useful parameters using conventional BLP-parameter-learning approaches such as EM (Kersting & De Raedt, 2008) have resulted in limited success as seen in the previous chapter. Consequently, we propose an alternate approach to specifying parameters for the learned first-order rules using lexical information from a curated ontology such as WordNet (Fellbaum, 1998). The basic idea behind our approach is that more accurate rules typically have predicates that are closely related to each other in terms of the meanings of the English words used to name them. Since WordNet is a rich resource for lexical information, we propose to use it for scoring rules based on word similarity.

The main contributions of this chapter are as follows:

- A novel online rule learner that efficiently learns accurate rules from noisy and incomplete natural-language extractions.
- A novel approach to scoring such rules using lexical information from a curated ontology such as WordNet (Fellbaum, 1998).

The rest of the chapter is organized as follows. In Section 5.2, we describe our online rule learner and in Section 5.3, we describe our novel approach to scoring rules. In Section 5.4, we present our experimental methodology and discuss results. We discuss related work in Section 5.5 and conclude in Section 5.6.

5.2 Online Rule Learner

In this section, we describe our online rule learner for inducing probabilistic first-order rules from the output of an off-the-shelf IE system. It involves construct-

Algorithm 2 Online Rule Learner

Inputs: Training examples D, target predicates T, and number of rules to output per predicate n. Each example D_i consists of a set of extractions.

Output: First-order definite clauses R for target predicates T.

for each example D_i **do**
  for each extraction x in D_i **do**
    Get the predicate P_x for x
    if P_x is a relation predicate **then**
      Increment count for P_x
    end if
  end for
  Construct a directed graph G_i in which relation extractions are nodes.
  for each pair of relations x and y that share one or more constants **do**
    Let P_x be the predicate of x and P_y be the predicate of y
    if count of $P_y <$ count of P_x **then**
      Add an edge from x to y
    end if
  end for
  for each relation x in the directed graph **do**
    for each outgoing edge (x,y) from x **do**
      Let y be the head node of the edge
      Create a rule R_j $x \rightarrow y$
      for each constant c_k in x **do**
        Add the type corresponding to c_k to the body of R_j
      end for
      Replace all constants in R_j with unique variables to create a first-order rule FR_j
      if FR_j is range restricted **then**
        Add FR_j to R and update the support for FR_j
      end if
    end for
  end for
end for
Sort rules in the descending order of their support and output top n rules for each predicate.

ing a *directed graph* of relation extractions for each training example and connecting those relations that share one or more constants with a directed edge. In the directed graph, each node represents a relation literal. A directed edge between two nodes indicates that the corresponding relations might be related, and hence might participate in the same rule. The edges are added from relations that are usually explicitly stated in text to those that can be inferred. Since relations that are implicit typically occur less frequently in the training data, we use the frequency of occurrence of relation predicates as a *heuristic* to determine if a particular relation is best inferred from other relations. While this assumption does not hold for all relations, it typically produces a useful set of first-order rules. We also note that our approach learns rules from only positive instances, thereby making it suitable for learning first-order rules for implicit relations.

The pseudocode for our Online Rule Learner (ORL) is shown in Algorithm 2. It accepts a set of training examples, where each example consists of a set of facts an IE system has extracted from a single document. The learner processes one example at a time in an online manner as follows. First, it updates counts for the frequency of occurrence for each relational predicate seen in the training example. The count for each relational predicate is the number of times it is seen in the training set. Then, it builds a directed graph whose nodes represent relation extractions seen in the example. Note that entity types are not added to the graph. The rule learner then adds *directed* edges between every pair of nodes whose relations share one or more constants as arguments. The direction of the edge is determined as follows – for every pair (x,y) of relations that share constants, if the relation

predicate of x is seen more frequently than that of y in the training set so far, then the learner adds a directed edge from x to y since y is more likely to be inferred from x. Note that the *for* loop in line 9 loops over both orders of each pair, i.e. both (x,y) and (y,x) will be considered. We also note that if the count of relation predicate of x is equal to the count of relation predicate of y, the algorithm does nothing.

Once the directed graph is fully constructed, the rule learner traverses the graph to construct rules. For each directed edge (x,y) in the graph, it constructs a rule in which the body contains x and y is the head. It then adds types corresponding to the constants in x. If a constant is associated with multiple types, i.e. if the extractor has extracted multiple types for a constant, then we create a separate rule for each type extracted for the constant. This is needed in domains that have a DAG-structured ontology as described earlier. Finally, it replaces all constants in the rule with unique variables to create a first-order rule. All first-order rules that are range restricted (all variables in the head appear in the body) are retained and the remaining rules are discarded.

The training phase ends when the rule learner has processed all examples in the training set. It then outputs the top n rules per predicate, where n is a value provided by the user. The rules are sorted in descending order of their *support*, which refers to to the number of times both the body (antecedent) and the head (consequent) in the rule are true in the training set. Alternately, the rule learner could output only those rules whose support meets a user-specified threshold.

In the basic algorithm, we have considered rules in which the body of the

rule has a *single* relation literal. However, we can extend the algorithm in several ways to search for rules that have several relation literals in them. For instance, given two rules $A \rightarrow B$ and $C \rightarrow B$, the rule learner can propose a new rule $A \wedge C \rightarrow B$. An alternate approach would be to follow the directed edges for a given path length and add all relations except that corresponding to the end node in the path to the rule body and make the relation corresponding to the end node the head. We found that the basic algorithm worked well for our application domain and hence we learned rules with a single relation in the rule body. Note that the rule body has other literals specifying the types of the arguments, and hence the rule bodies are not limited to have a single literal in them.

In some ways, the rules learned by this approach are similar to the typed entailment rules considered by Berant et al. (2011). However, as described above, unlike their approach, our method is not limited to learning rules with a single relation in the body. Furthermore, approaches that learn typed entailment rules like Berant et al.'s do not handle DAG ontologies in which relations can take arguments of multiple types. On the other hand, our method explicitly handles this situation.

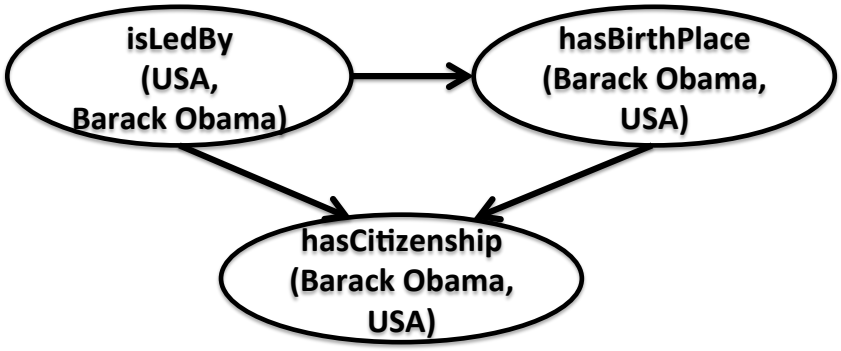
Consider the example shown in the Figure 5.1. Figure 5.1a shows a sample training document and Figure 5.1b shows the corresponding IE output. Given these extractions and the frequency counts for relation predicates seen so far in training (Figure 5.1c), our ORL algorithm constructs a directed graph with relations as nodes (Line 8 in Algorithm 2). It then adds directed edges between nodes that share one or more constants. In the example, relation extractions *isLedBy(usa,barack obama)*, *hasBirthPlace(barack obama,usa)*, and *hasCitizenship(barack obama,usa)*

(a) *Example text in training*
"Barack Obama is the 44th and the current President of USA... Obama, citizen of USA was born on August 4, 1961 in Hawaii, USA."

(b) *IE extractions*
nationState(usa)
person(barack obama)
isLedBy(usa,barack obama)
hasBirthPlace(barack obama,usa)
hasCitizenship(barack obama,usa)

(c) *Frequency counts for relation predicates*
isLedBy – 30, hasBirthPlace – 23, hasCitizenship – 20

(d) *Directed graph constructed from extracted relations*



(e) *Ground rules constructed by ORL*
isLedBy(usa,barack obama) ∧ person(barack obama) ∧ nationState(usa)
→ hasBirthPlace(barack obama,usa)
isLedBy(usa,barack obama) ∧ person(barack obama) ∧ nationState(usa)
→ hasCitizenship(barack obama,usa)
hasBirthPlace(barack obama,usa) ∧ person(barack obama) ∧ nationState(usa)
→ hasCitizenship(barack obama,usa)

(f) *First order rules constructed by ORL*
isLedBy(X,Y) ∧ person(Y) ∧ nationState(X) → hasBirthPlace(Y,X)
isLedBy(X,Y) ∧ person(Y) ∧ nationState(X) → hasCitizenship(Y,X)
hasBirthPlace(X,Y) ∧ person(X) ∧ nationState(Y) → hasCitizenship(X,Y)

Figure 5.1: Sample example describing various stages of the ORL algorithm

are added as nodes. Note that the entities *nationState(usa)* and *person(barack obama)* are not added to the graph. Since all relation extractions share constants *barack obama* and *usa*, ORL adds directed edges between them. The direction is determined by the frequency counts of relation predicates *isLedBy*, *hasCitizenship*, and *hasBirthPlace* as described in Lines 9–14 in Algorithm 2. For instance, since *isLedBy* is seen more often than *hasBirthPlace* in training, *hasBirthPlace* is more likely to be inferred from *isLedby*. Hence, the rule learner adds a directed edge from *isLedBy(usa,barack obama)* to *hasBirthPlace(barack obama,usa)*. After constructing the graph, ORL constructs rules as described in Lines 16–21. Finally, it replaces constants *barack obama* and *usa* with variables to construct first-order rules (Lines 22). Since all three rules are range restricted, they are not discarded.

5.3 Scoring rules using WordNet

We now discuss our new approach to determining parameters for the learned first-order rules. Our approach learns weights between 0 and 1, where a higher weight represents higher confidence. These weights are used as noisy-or parameters when performing probabilistic inference in the resulting BLP (Kersting & De Raedt, 2008; Raghavan et al., 2012). Since the predicate names in most ontologies employ ordinary English words, we hypothesized that more confident rules have predicates whose words are more semantically related. We use the lexical information in WordNet (Fellbaum, 1998) to measure word similarity.

WordNet is a lexical knowledge base covering around 130,000 English words in which nouns, verbs, adverbs, and adjectives are organized into synonym sets, also

called *synsets*. Several measures (Resnik, 1995; Wu & Palmer, 1994; Lin, 1998) have been proposed to measure word similarity based on the semantic information in WordNet. Of these measures, the *wup* measure by Wu and Palmer (1994) computes (scaled) similarity scores between 0 and 1, which are easily used as weights for our rules. Therefore, we used *wup* as implemented in WordNet::Similarity (Pedersen, Patwardhan, & Michelizzi, 2004) to measure semantic distances between words. This measure computes the depth of the least common subsumer (LCS) of the given words and then scales it by the sum of the depths of the given words.

We compute the *wup* similarity for every pair of words (w_i,w_j) in a given rule, where w_i is a word in the body and w_j is a word in the head. The words in a given rule are the predicate names of relations and entity types, which are usually English words. However, for predicate names such *hasCitizenship* or *hasMember* that are not single English words, we segment the name into English words such as *has*, *citizenship*, and *member*, and then remove stop words. The final weight for a rule is the average similarity between all pairs (w_i,w_j), which basically measures how closely predicates in the body are related to the predicate in the head. We refer to this approach as "WUP-AVG".

Figure 5.2 gives sample rules and the corresponding English words for the predicate names. The values in parentheses give the weights computed using WUP-AVG for the corresponding rule using the approach described above. Notice that in Rule 1, *governmentOrganization* is segmented into words *government* and *organization*. Similarly, *hasMember* is segmented into *has* and *member* and the stopword *has* is removed while computing WUP-AVG. Similarly, other predicates in the rule

Rule 1
employs(X,Y) $\wedge$ governmentOrganization(X) $\rightarrow$ hasMember(X,Y) (.70)
English words for predicate names
(employs, government, organization) $\rightarrow$ (member)
Rule 2
employs(X,Y) $\wedge$ person(Y) $\wedge$ nationState(X) $\rightarrow$ hasBirthPlace(Y,X) (.67)
English words for predicate names
(employs, person, nation, state) $\rightarrow$ (birth, place)

Figure 5.2: Example rules and the corresponding English words for the predicates in the rule. Rule weights computed using WUP-AVG are shown in the parentheses.

are processed. We see that the weight computed for Rule 1 is higher than that computed for Rule 2, since Rule 1 is more likely to be true than Rule 2. Here, the absolute weights computed are less important as the relative ordering of the rules impact the final ranking of the inferences.

We also explored alternate approaches for computing rule weights using the *wup* score. Instead of computing the average similarity between all pairs of words in the body and the rule head, we used the highest similarity score among all word pairs as the weight. We refer to this as "WUP-MAX". In an alternate approach, we used the highest similarity score among all word pairs from relation predicates only, i.e. we did not take into account words from entity types while computing the similarity score. We refer to this as "WUP-MAX-REL".

5.4 Experimental Evaluation

5.4.1 Data

We evaluated our approaches to rule learning and scoring on DARPA's machine-reading intelligence-community (IC) data set. As described in Chapter 4, the IC dataset consists of news articles on terrorist events around the world. The data set consists of $10,000$ documents, each containing an average of 93.14 facts extracted by SIRE (Florian et al., 2004), an IE system developed by IBM[1].

As described earlier, the ontology provided by DARPA for the IC domain consists of 57 entity types and 79 relations. The entity types include *Agent*, *PhysicalThing*, *Event*, *TimeLocation*, *Gender*, and *Group*, each with several subtypes. The type hierarchy is a DAG rather than a tree, and several types have multiple super-classes. For instance, a *GeopoliticalEntity* can be a *HumanAgent* as well as a *Location*. Relations in the ontology include *eventLocation*, *thingPhysicallyDamaged*, *attendedSchool*, and *hasCitizenship*.

5.4.2 Evaluation Measure

We used the same methodology that was used in Chapter 4 for evaluation. We randomly sampled 4 documents from each test set, 40 documents in total. We manually evaluated the inferences since there is no ground truth available for this data set. We ranked all inferences in descending order of their marginal probabilities and computed precision for top n inferences. The precision measures the

[1]In Chapter 4, we reported a different number for the average number of extractions per document due to differences in the preprocessing of the dataset.

employs	eventLocation
eventLocationGPE	hasMember
hasMemberPerson	isLedBy
mediatingAgent	thingPhysicallyDamaged
hasMemberHumanAgent	killingHumanAgent
hasBirthPlace	thingPhysicallyDestroyed
hasCitizenship	attendedSchool

Table 5.1: Target relations selected for experimental evaluation

fraction of inferences that were judged correct. As described in the previous chapter, we computed two different precision scores - unadjusted (UA) and adjusted (AD) precision.

5.4.3 Evaluation of Online Rule Learner

We evaluated our approach by performing 10-fold cross validation. We learned first-order rules using 14 target relations given in Table 5.1 that had an appreciable amount of data. We describe the systems compared in our experimental evaluation below:

- ORL - We learn rules using our online rule learner described in Section 5.2. For each target relation, we specify the number of rules to output to be 10. We refer to this approach as "ORL". Table 5.2 gives sample rules learned by ORL along with rule weights computed using the approach described in Section 5.3.
- LIME - We used LIME (McCreath & Sharma, 1998), the ILP based rule learner that was used in Chapter 4 as a baseline for comparison. As described

in Chapter 4, we learned rules using only positive instances and using both positive and negative instances for each target relation. Since the IC data set consists of only positive instances for target relations, the negative instances were artificially generated using the closed world assumption. The final structure included rules learned from both settings. We refer to this baseline as "LIME". Note that we cannot compare our results to those reported in Chapter 4 due to the differences in the number of target predicates selected for the study. Also, in the previous chapter (Chapter 4), we used extractions whose confidence scores exceeded a certain threshold for learning rules. Here, we use all extractions to learn first-order rules since the approach used in the previous chapter resulted in smaller training sets for learning weights using EM.

- COMBINED - Both ORL and LIME learn some rules that the other doesn't, and hence we combined rules from both approaches in this final setting, which we refer to as "COMBINED".

We observed that all methods learn inaccurate rules for certain target relations such as *mediatingAgent* and *attendedSchool* since they are less easily inferred compared to other relations such as *hasMember* that are more easily inferred. Therefore, we removed 4 relations – *mediatingAgent*, *attendedSchool*, *thingPhysicallyDamaged*, and *thingPhysicallyDestroyed* from the original set and also report results for the remaining 10 target relations. We refer to the original set of target relations as "Full-set" and the reduced set as "Subset". For all three approaches described above, we

isLedBy(B,A) ∧ person(A) ∧ nationState(B) → hasBirthPlace(A,B) (0.62) *If person A leads a nation B, then A is born in B*
thingPhysicallyDamaged(A,B) ∧ bombing(A) ∧ nationState(B) → eventLocation(A,B) (0.71) *If a nation B is physically damaged in a bombing event A, then the event location of A is B*
employs(A,B) ∧ humanOrganization(A) ∧ personGroup(B) → hasMemberHumanAgent(A,B) (0.57) *If a human organization A employs B, then B is a member of A*
isLedBy(B,A) ∧ nationState(B) → hasCitizenship(A,B) (0.48) *If a nation B is led by A, then A is a citizen of B*

Table 5.2: A sample set of rules learned by ORL

	Full-set	Subset
ORL	101.8	85.2
LIME	58.7	50.7

Table 5.3: Average number of rules learned per fold by LIME and ORL

run BLP inference using Full-set and Subset. Table 5.3 gives the average number of rules learned per fold by both LIME and ORL for both Full-set and Subset.

5.4.3.1 BLP Parameters and Inference

As described in previous chapters (see Chapter 2 and Chapter 4), we used the deterministic *logical-and* model to encode the CPT entries for the Bayesian clauses and the *noisy-or* model to combine the evidence from multiple rules that have the same head (Pearl, 1988). As seen in Chapter 4, learning noisy-or parameters for the learned rules using the EM algorithm developed for BLPs (Kersting & De Raedt, 2008) resulted in limited success. As a result, we manually set the

noisy-or parameters for all rules to 0.9, since these parameters have been shown to work well on the IC domain (see Chapter 4). To infer implicit facts, we perform inference in the BLP framework as described in Chapter 4. We used SampleSearch (Gogate & Dechter, 2007) to compute marginal probabilities for the inferred facts.

5.4.3.2 Results

Figure 5.3 and Figure 5.4 give unadjusted and adjusted precision for inferences made by rules learned using ORL, LIME, and COMBINED models for target relations from both Full-set and Subset respectively. On the Full-set, LIME outperforms ORL at the initial points on the curve, while ORL outperforms LIME by a significant margin at the later points on the curve. However, on the Subset, ORL outperforms LIME at all points in the curve. A closer examination of the rules learned by both approaches revealed that ORL learned rules that were more specific than LIME. As a result, ORL makes far fewer but more accurate inferences than LIME. We observed that the curve for ORL stops at top-900 and top-500 for Full-set and Subset respectively. This is because ORL does not make more than 900 and 500 inferences on Full-set and Subset respectively. On both Full-set and Subset, COMBINED outperforms both ORL and LIME on both settings, indicating that there are definite advantages to combining rules from both LIME and ORL. For the remaining experiments, we run inference using the COMBINED model only. In general, performance of all models on the Subset is better than that on the Full-set.

On both Full-set and Subset, we find that the precision does not monotonically decrease as n, the number of inferences considered for evaluation increases.

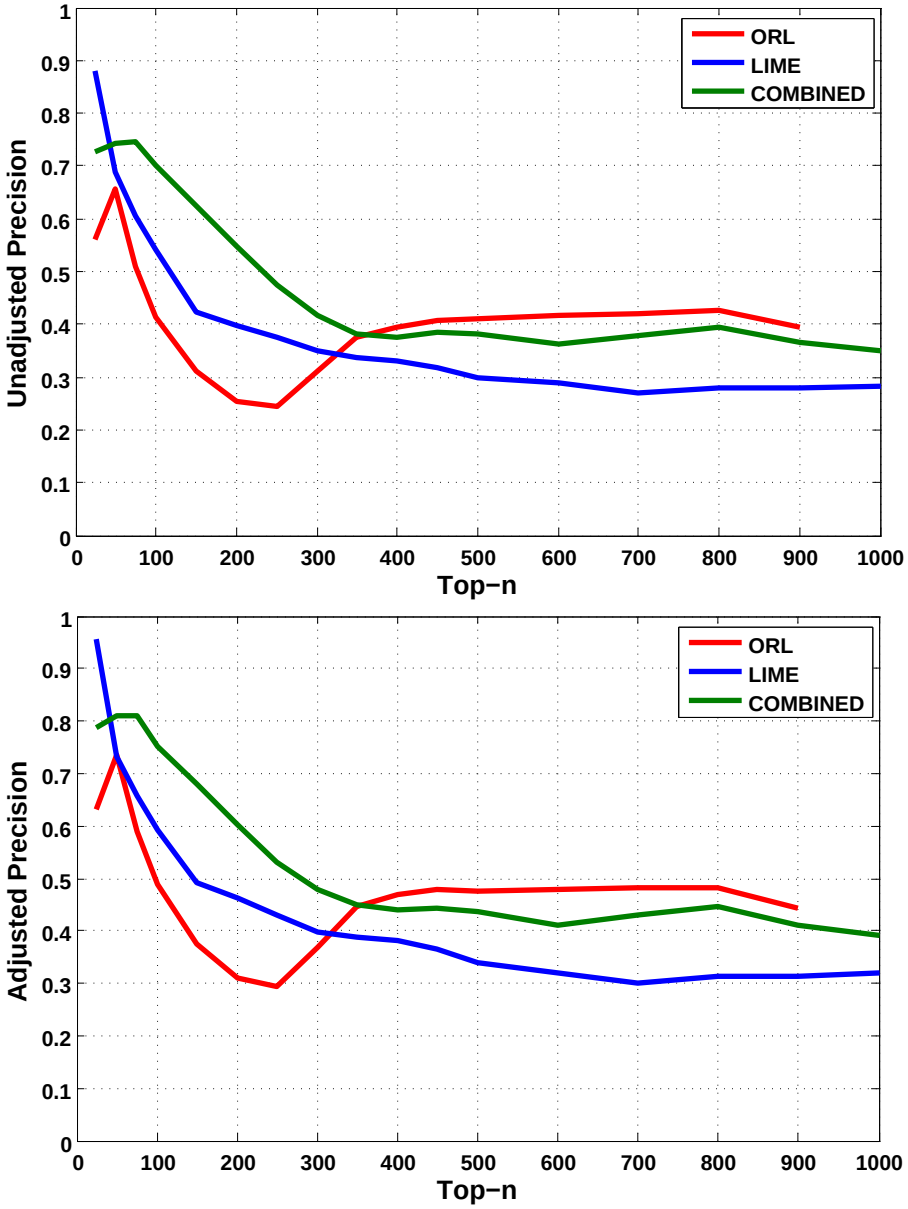


Figure 5.3: Unadjusted (top) and adjusted (bottom) precision at top-n for ORL, LIME, and COMBINED for target relations from Full-set

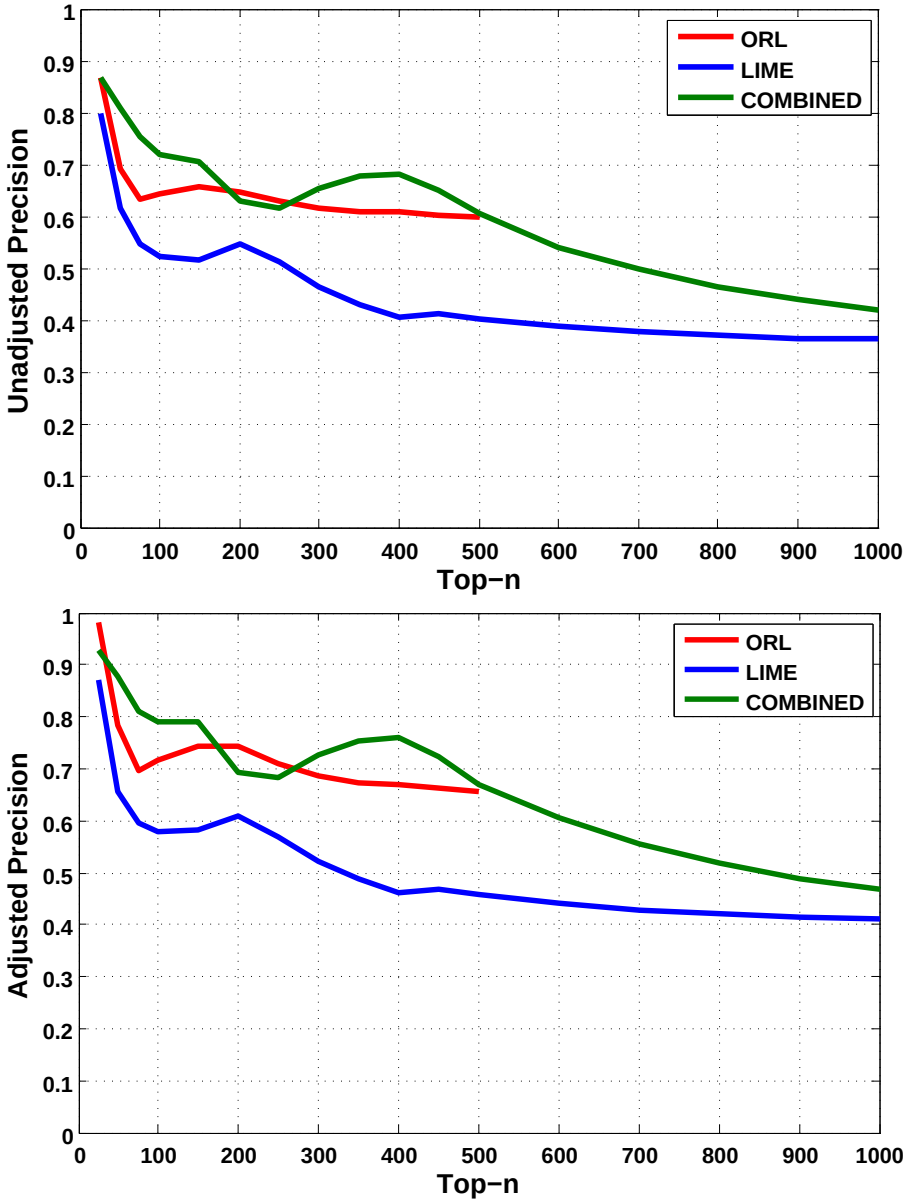


Figure 5.4: Unadjusted (top) and adjusted (bottom) precision at top-n for ORL, LIME, and COMBINED for target relations from Subset (bottom)

We find that several incorrect inferences are ranked higher since they have higher marginal probabilities due to multiple rules inferring the same instance. On the other hand, several correct inferences are ranked lower due to lower marginal probabilities, possibly due to lower evidence in the form of a single rule inferring the instance. Since all rules have the same weight (noisy-or parameter), facts inferred from multiple rules typically end up with higher marginal probabilities than those that are inferred from a single rule. We hypothesize that learning noisy-or parameters automatically using sufficient training data might alleviate such problems.

We also notice that even though the COMBINED model includes rules from both LIME and ORL, the performance of the COMBINED model does not necessarily reflect the combined performance of the underlying model. This is because by combining rules from two different models, we are defining a new structure for the BLP. The rules in the COMBINED model interact in a different manner from those in the underlying models (ORL and LIME). Further, in the COMBINED model, several target relations might have many more rules defining them than in LIME and ORL. If several of these rules are responsible for inferring the same fact, then the marginal probability for this inference might be higher due to higher evidence, and hence this inference might be ranked higher in the COMBINED model than in LIME or ORL. If this inference is deemed correct, then the performance of the COMBINED model might increase, otherwise it might decrease at different top-n. As a result, we do not expect the performance of COMBINED model to be a linear combination of that of LIME and ORL.

The noisy-or parameters in the current experiments were set to 0.9. We

also ran inference on the COMBINED model by setting the noisy-or parameters to alternate values, 0.5 and 0.75. The performance with alternate weights remained the same, indicating that the actual value of the noisy-or parameter did not play a crucial role since the relative order of the ranked inferences remained the same. The ranked order of the inferences is typically determined by the number of rules that entail each inference.

In the next experiment, we evaluated all inferences made by LIME, ORL, and COMBINED without ranking them based on their marginal probabilities. It is equivalent to performing inference using purely logical deduction using the rules learned by different models. Table 5.4 gives both adjusted and unadjusted precision for target relations from Full-set and Subset. We find that the inferences made by ORL are more accurate than those made by LIME or the COMBINED model. However, ORL makes fewer inferences than LIME and COMBINED. Here again, precision on the Subset is higher than that on the Full-set. These results indicate that the rules learned by ORL are fairly accurate even though they are very specific. Since the rules are less general, the number of inferences made by ORL are fewer than those made by other models.

Table 5.5 gives unadjusted precision per relation for instances inferred at 0.95 confidence threshold by different models. We do not report adjusted precision since it is slightly higher than unadjusted precision. LIME is able to infer relations such as *employs* more accurately than ORL, while ORL is able to infer relations such as *hasBirthPlace* and *hasCitizenship* more accurately than LIME. Both LIME and ORL are able to infer relations such as *hasMember*, *hasMemberPerson*, and

	Full-set		Subset	
	UA	AD	UA	AD
LIME	24.14 (594/2460)	27.58 (594/2153)	33.57 (414/1233)	37.60 (414/1101)
ORL	**39.41** (361/916)	**44.18** (361/817)	**58.64** (329/561)	**64.00** (329/514)
COMBINED	18.73 (662/3533)	20.61 (662/3212)	38.17 (439/1150)	42.09 (439/1043)

Table 5.4: Precision for logical deduction using rules learned from LIME, ORL, and COMBINED. "UA" and "AD" refer to the unadjusted and adjusted scores respectively

hasMemberHumanAgent with fairly high precision. On the other hand, relations such as *isLedBy*, *mediatingAgent*, *thingPhysicallyDamaged* have low precision for both LIME and ORL. The COMBINED model has superior precision for all those relations for which either LIME or ORL have a high precision, thereby demonstrating a definite advantage over LIME and ORL.

In our final set of experiments, for each target relation, we eliminated all instances of it from the set of extracted facts in the test examples in each fold. We then ran the BLP inference using the remaining facts to deduce additional facts. We evaluated if the eliminated instances were inferred by the BLP approach at various levels of confidence (probability threshold). We measure the fraction of eliminated instances that were inferred correctly by the different models – LIME, ORL, and COMBINED. We refer to this measure as "estimated recall". In Chapter 4, we discussed the difficulty involved in measuring the *actual* recall for different models. While the estimated recall measure does not necessarily capture the true recall of a model, it definitely helps distinguish different models in their ability to infer elim-

	LIME	ORL	COMBINED
employs	71.87 (23/32)	nil (0/0)	72.72 (24/33)
eventLocation	5.00 (3/60)	27.78 (5/18)	9.64 (22/228)
eventLocationGPE	20.71 (29/140)	41.17 (14/34)	15.56 (47/302)
hasMember	86.84 (33/38)	85.71 (18/21)	85.21 (98/115)
hasMemberPerson	80.00 (8/10)	100 (1/1)	57.44 (27/47)
isLedBy	0.00 (0/39)	0.00 (0/16)	7.60 (7/92)
mediatingAgent	4.16 (1/24)	nil (0/0)	3.03 (1/33)
thingPhysicallyDamaged	9.40 (11/117)	8.75 (14/160)	9.18 (26/283)
hasMemberHumanAgent	60.71 (17/28)	86.67 (13/15)	81.92 (68/83)
killingHumanAgent	12.50 (11/88)	nil (0/0)	12.94 (11/85)
hasBirthPlace	38.46 (40/104)	75.00 (39/52)	36.43 (47/129)
thingPhysicallyDestroyed	nil (0/0)	nil (0/0)	0.00 (0/4)
hasCitizenship	39.02 (32/82)	77.19 (44/57)	40.98 (50/122)
attendedSchool	nil (0/0)	nil (0/0)	nil (0/0)

Table 5.5: Unadjusted precision for individual relations

inated instances. Further, these experiments allow for automatic evaluation since the ground truth is available in the form of instances that are eliminated.

In these experiments, it is possible that certain eliminated instances can never be inferred by any of the models because the facts necessary to infer the eliminated instance are not present in the remaining extractions. However, if there is sufficient redundant information in the extracted facts to infer an eliminated instance, then these experiments help distinguish different models in their ability to infer these eliminated instances. For example, suppose we randomly eliminated a sentence from a document. Sometimes, due to redundancy in natural language text, it is possible to infer the sentence based on the other information present in the text. However, usually, it is not possible to infer this sentence at all. Similarly, in our context, some instances of the target relation can never be inferred, so we might

see low estimated recall scores. It is the difference in the estimated recall scores calculated for different models that help distinguish their relative performance.

Figure 5.5 shows the estimated recall, averaged over all target relations on both Full-set and Subset for different marginal probability thresholds. As the confidence level or the marginal probability threshold increases, fewer instances whose marginal probabilities meet the threshold are inferred; as a result, the estimated recall goes down. LIME performs better than ORL, while COMBINED performs better than both LIME and ORL. Here again, the superior performance of COMBINED is due to combining rules from both ORL and LIME, which typically results in the inference of larger number of facts than LIME and ORL. As mentioned earlier, since the rules learned by LIME are less specific than those learned by ORL, LIME is capable of inferring more facts than ORL, which is possibly the reason for a higher estimated recall when compared to that for ORL. However, we note that the relative differences in the scores for different models are quite small. Here again, the estimated recall for different models on the Subset is marginally higher than that on the Full-set.

Figure 5.6, Figure 5.7, and Figure 5.8 show estimated recall per target relation at different marginal probability thresholds for a few target relations. For target relations *isLedBy* and *eventLocation*, ORL outperforms LIME while for target relations *killingHumantAgent* and *thingPhysicallyDamaged*, LIME outperforms ORL. For the remaining target relations, both ORL and LIME performed similarly with respect to estimated recall and hence we do not show the results for them here. Here again, COMBINED outperforms ORL and LIME demonstrating defi-

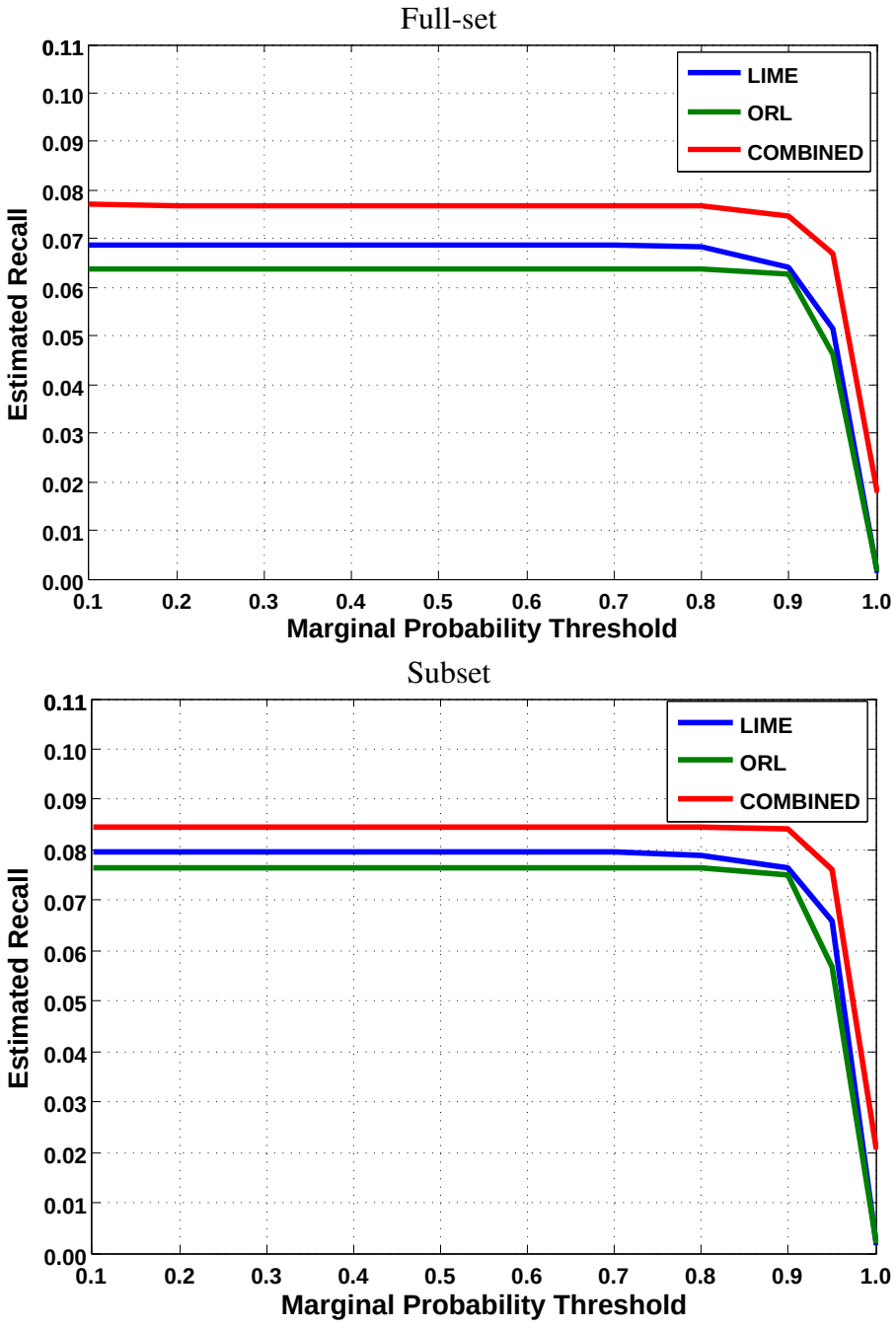


Figure 5.5: Estimated recall at different levels of confidence on Full-set (top) and Subset (bottom)

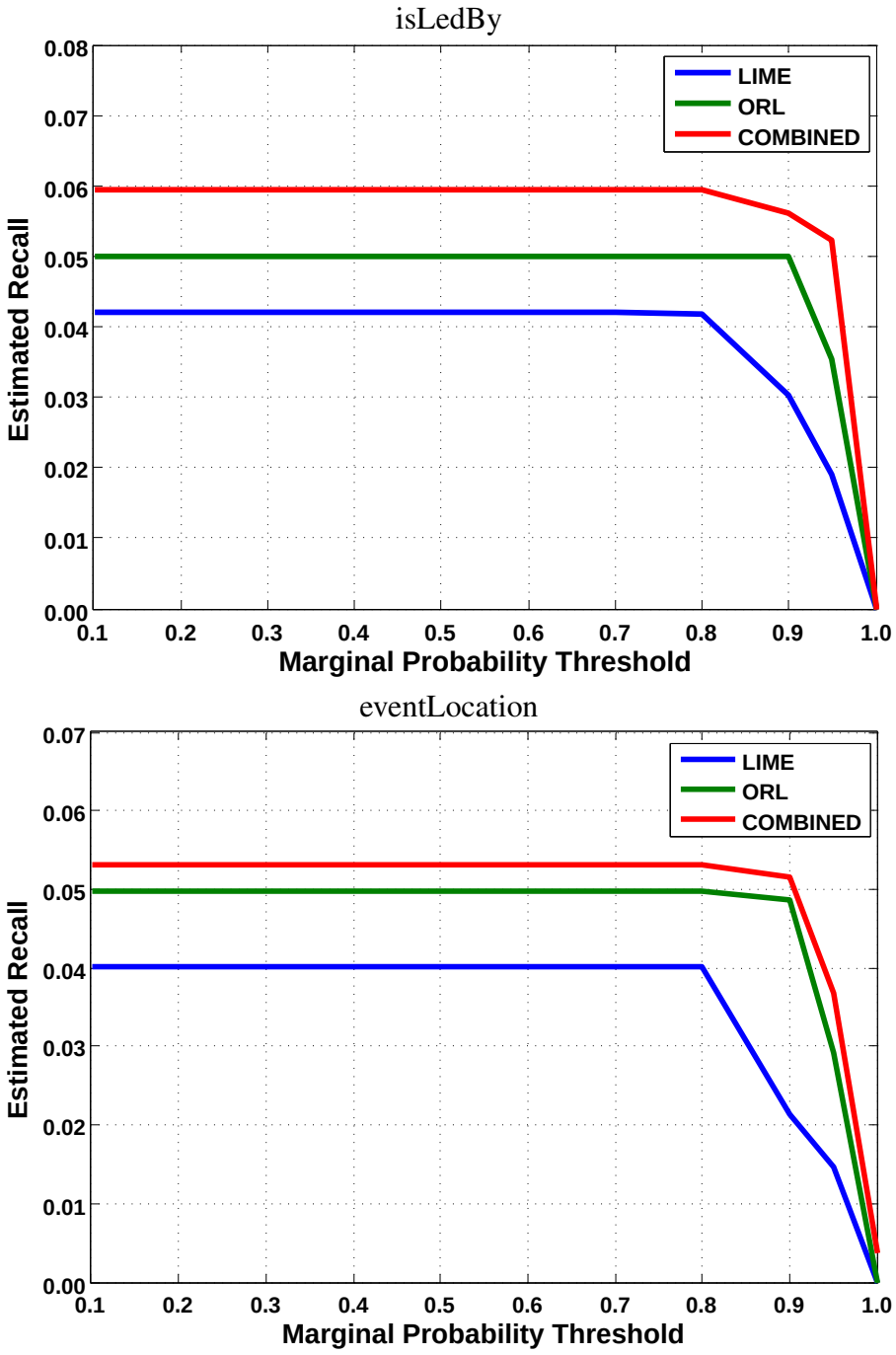


Figure 5.6: Estimated recall at different levels of confidence for relations *isLedBy* (top) and *eventLocation* (bottom)

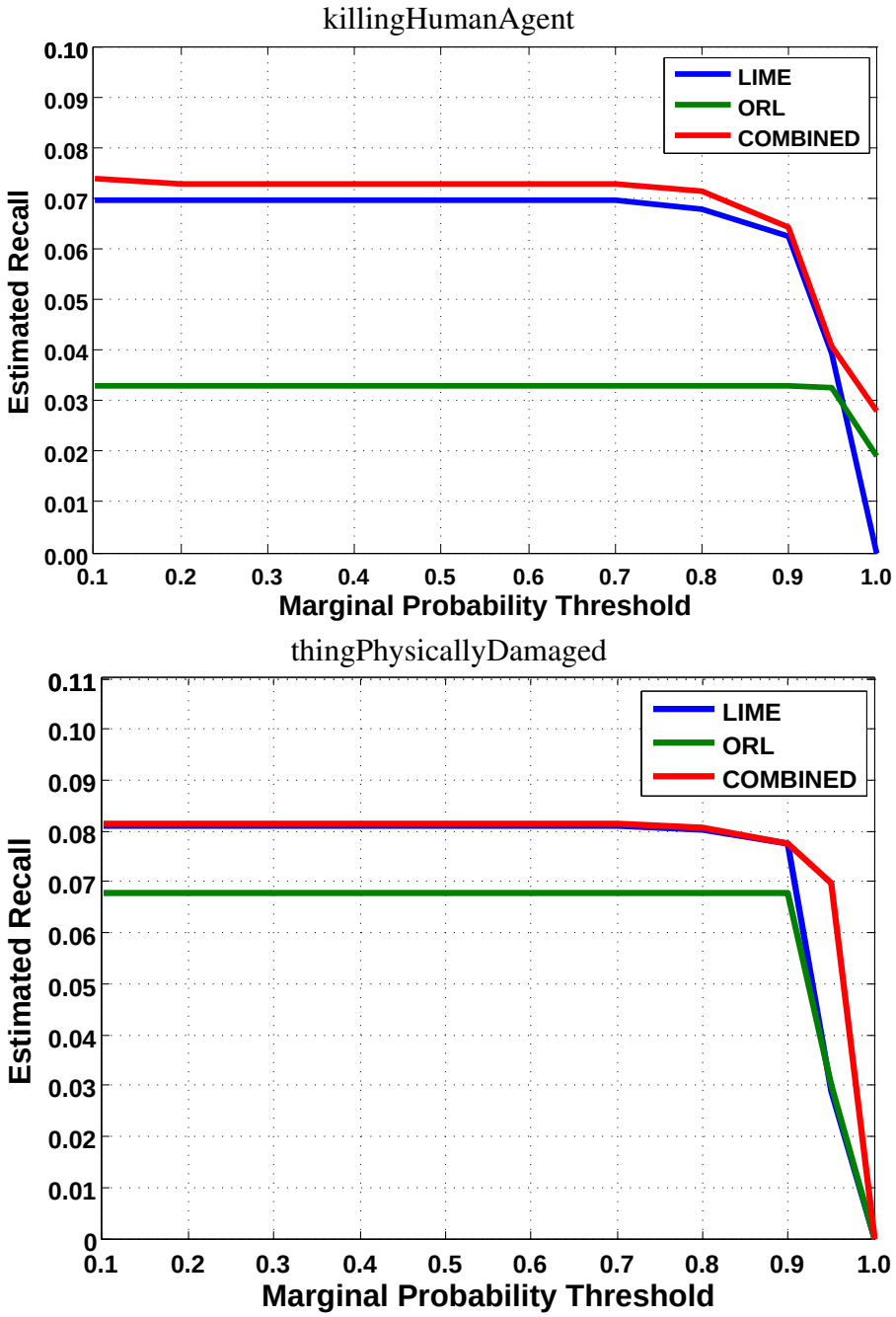


Figure 5.7: Estimated recall at different levels of confidence for relations *killingHumanAgent* (top) and *thingPhysicallyDamaged* (bottom)

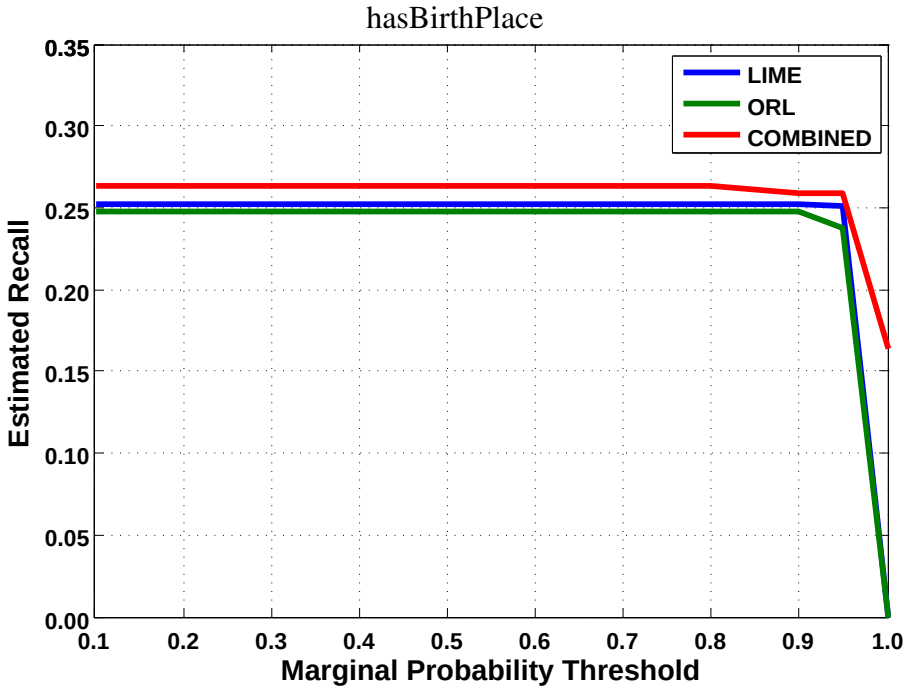


Figure 5.8: Estimated recall at different levels of confidence for relation *hasBirthPlace*

ORL	LIME
3.8	673.98

Table 5.6: Average training time per fold in minutes

nite advantages to combining rules from both ORL and LIME. For target relation *hasBirthPlace*, all models have similar performance. However, this is an example of a target relation for which the estimated recall scores are much higher than those on the remaining target relations. Results for *hasBirthPlace* indicate that the eliminated instances of *hasBirthPlace* are easily inferred, possibly due to the presence of redundant information in the form of extracted facts in the test documents. The lower estimated recall scores for other target relations indicate that there might not be sufficient additional information that is necessary for the inference of the eliminated instances.

Table 5.6 gives the average time in minutes per fold needed to learn rules using both ORL and LIME. As discussed is Chapter 4, since LIME does not scale to large data sets, we ran LIME on smaller subsets of the training data and combined the results in order to process the full IC data. The runtime for LIME includes the total time taken to produce the final set of rules. Unlike ORL, LIME learns first-order rules for each target predicate separately, further increasing its running time. On the other hand, ORL learns rules for all target predicates in one pass on the training data. As a result, ORL trains two orders of magnitude faster than LIME. The timing information empirically demonstrates ORL's ability to scale effectively to large data sets.

Target Relation	Time in minutes
eventLocation	24.11
thingPhysicallyDestroyed	9.11
mediatingAgent	330.10
isLedBy	42.54
hasCitizenship	7.76
thingPhysicallyDamaged	114.50
hasMember	10.04
attendedSchool	6.34
hasMemberHumanAgent	12.96
employs	75.22
eventLocationGPE	10.15
hasBirthPlace	7.46
killingHumanAgent	15.28
hasMemberPerson	8.33

Table 5.7: Average training time taken by LIME to learn first-order rules per fold for individual target relations.

Table 5.7 gives the time taken by LIME to learn first-order rules for individual relations. These running times include the time taken by LIME to learn rules using only positive instances and using both positive and negative instances. LIME takes longer time to learn first-order rules for relations such as *mediatingAgent* and *thingPhysicallyDamaged* due to the large size of the respective training sets. The training sets for these relations are much larger due to the large number of negative instances generated using the closed world assumption. For the remaining relations, LIME takes lesser time due to the smaller training set sizes. The time taken by LIME to learn rules from positive only instances is not very high. However, LIME learns fewer rules from positive only instances and these rules are very general, and hence less useful for the purposes of inferring additional facts from natural

language text. On the other hand, ORL learns fairly specific rules in a very short amount of time and these rules result in fairly accurate inferences.

5.4.4 Scoring rules using WordNet

We learned noisy-or parameters using the different approaches (WUP-AVG, WUP-MAX, and WUP-MAX-REL) described in Section 5.3 on the COMBINED model. For the baseline, we set all noisy-or parameters to 0.9 and performed inference in the BLP framework as described in Section 5.4.3.1, which we refer to as "Default". We also explored learning the noisy-or parameters using the EM algorithm as described in Chapter 4. While we were able to learn noisy-or parameters for target relations from Subset, we were unable to learn the parameters on the Full-set due to large network sizes and longer running times. Logical inference for several examples was also not tractable since the Full-set had a large number of rules. For target relations from the Subset, we also compare the performance of our rule scoring approach to noisy-or parameters learned using EM, which we refer to as "EM".

5.4.4.1 Results

Figure 5.9 and Figure 5.10 give the precision for inferences made by the COMBINED model using noisy-or parameters computed using different approaches on both Full-set and Subset respectively. WUP-AVG outperforms Default on both sets of target relations. WUP-MAX, on the other hand performs poorly when compared to both Default and WUP-AVG on the Full-set, but outperforms Default and

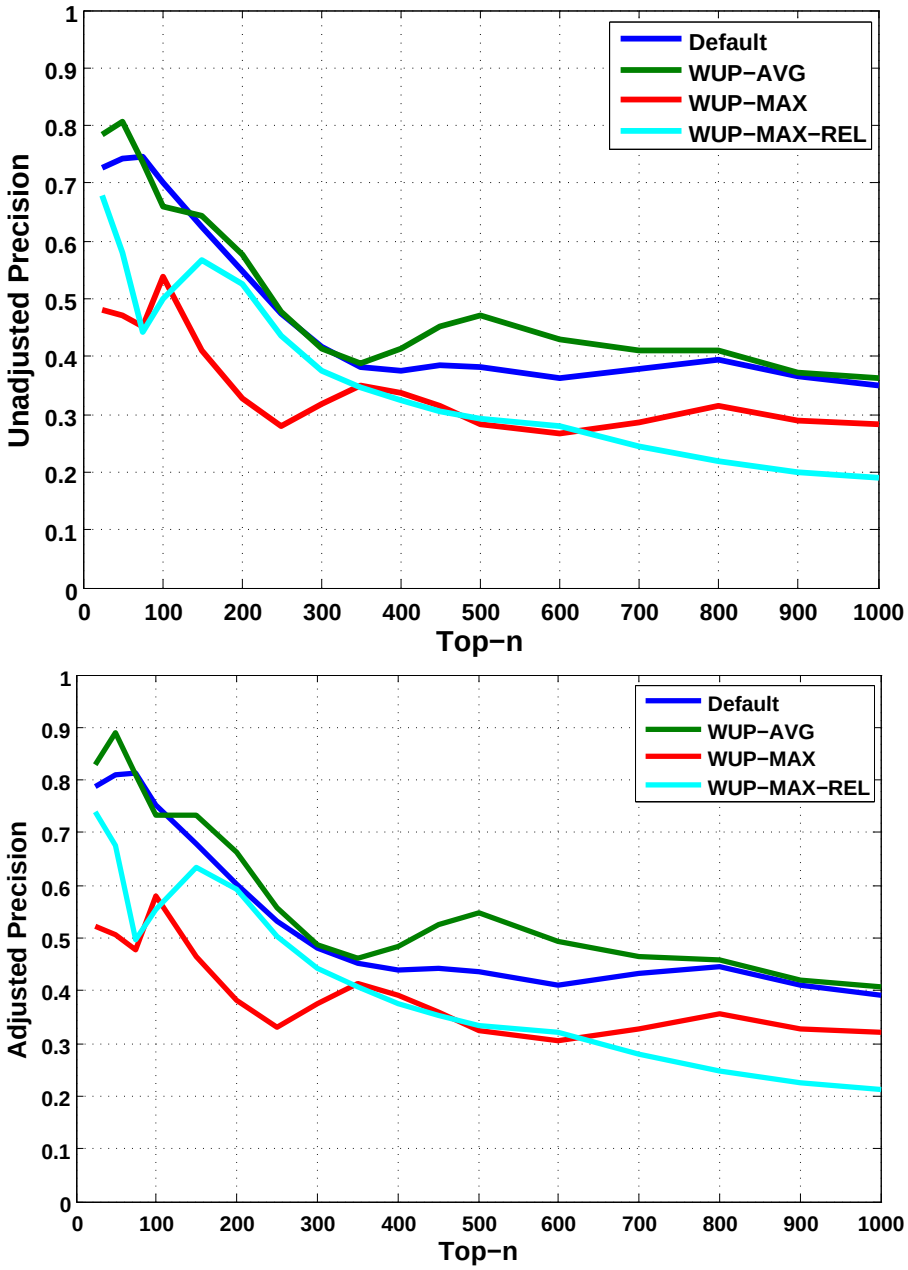


Figure 5.9: Unadjusted (top) and adjusted (bottom) precision at top-n using different weights on Full-set

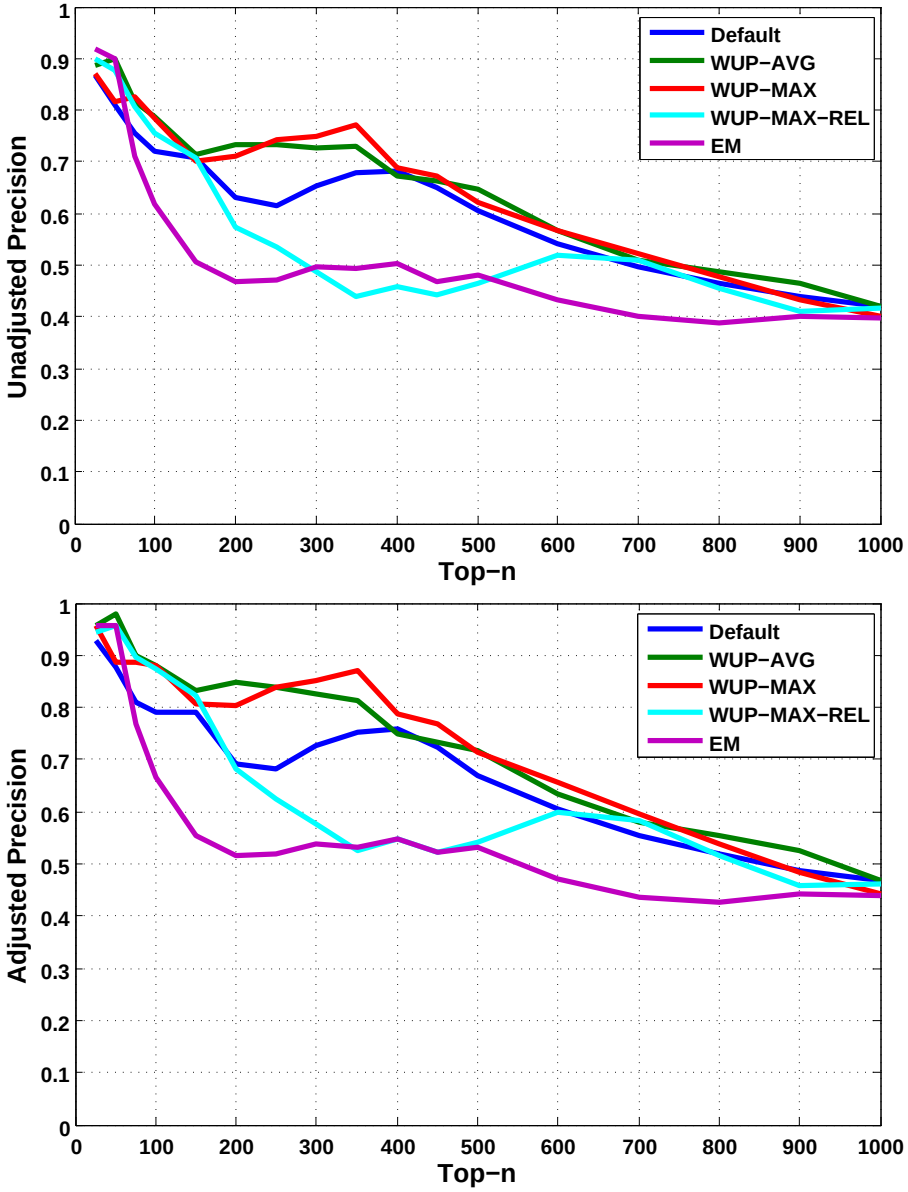


Figure 5.10: Unadjusted (top) and adjusted (bottom) precision at top-n using different weights on Subset

is competitive with WUP-AVG on the Subset. On both sets, WUP-MAX-REL is generally inferior to both Default and WUP-AVG. However, it is somewhat better than WUP-MAX on Full-set and is inferior to WUP-MAX on the Subset. On the Subset, most of these approaches are fairly competitive at top-25, however they exhibit dramatic differences in the performance at other points in the curve. Weights learned from EM are again competitive with weights learned from other approaches at top-25 on the Subset. However, at later points in the curve, the performance of EM drops drastically indicating that the weights learned by EM are inferior to those computed using other approaches. Lack of sufficient data is one of the main reasons for the poor performance of EM. The results seen for EM are very similar to the results reported in Chapter 4. As seen in the earlier results, performance on the Subset is better than that on the Full-set.

In general, we find that WUP-AVG is the best approach for computing rule weights. In the case of WUP-MAX, it is possible that one particular entity type or a relation predicate in the body of the rule could be very similar to the relation predicate in the head and that could dominate the final weight. Sometimes, it is even possible that the relation predicate in the body could be quite different from the one in the head, but the rule might get a high weight if one of the entity types in the body is similar to the relation predicate in the head. Consider the two rules shown in Table 5.8 along with the weights computed using WUP-MAX. The two rules shown in the example differ only by the entity type for the first argument. However, the weights computed are quite different demonstrating that the similarity between entity types and the relation predicate in the head dominate over other similarity

isLedBy(A,B) ∧ politicalParty(A) → hasMemberPerson(A,B) (0.91)
isLedBy(A,B) ∧ governmentOrganization(A) → hasMemberPerson(A,B) (0.86)

Table 5.8: A sample set of rules learned by ORL. Scores in parentheses are WUP-MAX scores.

scores. Here, the relation predicate in the body of the rule does not seem to play any role in determining the final weight. For all these reasons, WUP-MAX is not a very robust score for computing rule weights. The lack of robustness is also demonstrated by the large variation in its performance on the different set of target relations. While WUP-MAX-REL only considers the similarity between relation predicates, it can still be affected by some of the issues described above. WUP-AVG, on the other hand is more robust since it takes into account both relation predicates and entity types in the body of the rule while computing the similarity scores. When selecting an average of 1 inference per document, WUP-AVG gives an unadjusted precision of 80.6% and 90% on the Full-set and Subset respectively.

Overall, our online rule learning approach learns unique rules that are not learned by LIME. Unlike LIME, it not only scales to large corpora of natural language text, but also handles the concise, incomplete nature of natural-language text while learning first-order rules. Even though it does not always outperform LIME, combining rules from both approaches results in a model that gives the best performance on the IC data set. Though weight learning using conventional parameter learning techniques such as EM did not yield reasonable results, our approach to scoring rules using WordNet similarity scores has shown promise and it is an interesting topic for future research.

5.5 Related Work

Approaches proposed in the literature for learning first-order rules fall into three categories. The first category consists of rule learners (Quinlan, 1990; McCreath & Sharma, 1998; Kersting & De Raedt, 2008; Dinh et al., 2011) that typically expect the training data to be fairly accurate. Due to this assumption, they are less suited for learning meaningful rules from noisy and incomplete natural language extractions. The second category consists of rule learners (Schoenmackers et al., 2010; Sorower et al., 2011) that are specifically developed for learning rules from noisy and incomplete natural language extractions. Both these rule learners learn first-order rules for MLNs by performing an exhaustive search, which might be computationally intensive for large domains. Unlike our approach, Sorower et al. (2011) have evaluated their approach using only a couple of target relations. The third category consists of approaches (Lin & Pantel, 2001; Yates & Etzioni, 2007; Berant et al., 2011) that learn entailment rules to capture synonymous relations and entities from natural language text. Unlike the rules learned from the other learners, entailment rules have a single literal as the antecedent (body).

A few previous approaches have scored rules using lexical knowledge. Basu et al. (2001) use WordNet distances to estimate the novelty of rules discovered by data-mining systems. Feldman and Dagan (1995) and Han and Fu (1995) use domain-specific concept hierarchies to score and filter redundant rules. Garrette et al. (2011) use distributional lexical semantics to set weights for rules in an MLN.

Chapter 4 discusses approaches developed for inferring implicit facts from natural language text in detail. For the sake of completeness, we briefly discuss

them here as well. Nahm and Mooney (2000) learn propositional rules using C4.5 (Quinlan, 1993) from data extracted from computer-related job-postings. Carlson et al. (2010) and Doppa et al. (2010) use existing rule learners such as FOIL (Quinlan, 1990) and FARMER (Nijssen & Kok, 2003) to learn probabilistic first-order rules, and subsequently use those rules to infer additional information using purely logical deduction. Schoenmackers et al. (2010) and Sorower et al. (2011) develop novel rule learners to learn first-order rules from noisy natural language text and perform probabilistic inference in the MLN framework to infer implicit facts using the learned rules. We use LIME (McCreath & Sharma, 1998), an existing rule learner to learn first-order rules and then use the BLP framework to infer additional facts. The superior performance of BLPs over purely logical deduction and MLNs as demonstrated in Chapter 4 has led us to use BLPs for probabilistic inference in this chapter as well.

5.6 Summary

In this chapter, we have proposed a novel online rule learner for learning first-order rules from noisy and incomplete IE extractions, extracted using an off-the-shelf IE system. We have used the learned rules to infer implicit facts in natural language text using BLP inference. Experimental comparison of our rule learner with LIME, an existing rule learner demonstrates the efficacy of our approach. However, the best performance on the IC data set is obtained when rules from both approaches are combined. Further, we have also proposed a novel approach to specifying parameters for the learned rules based on lexical similarity between words or

relations using a curated ontology such as WordNet. Experimental evaluation on the IC data set demonstrates the superior performance of this new approach over using default weights set manually to 0.9 and weights learned using the EM algorithm. The initial promise demonstrated by our new approach to scoring rules reduces the dependency on conventional parameter learning approaches like EM, which are not very effective for learning parameters of BLPs on the machine reading task.

Chapter 6

Future Work

In this chapter, we discuss several directions in which we can extend our work described in this dissertation. We first discuss future work items that will lead to the improvement of BLP and BALP framework. These topics typically involve algorithmic improvements to fundamental techniques involving inference and learning. In the next section, we discuss topics specific to the task of plan recognition and machine reading that could be pursued in future.

6.1 Inference and learning in BLPs/BALPs

6.1.1 Structure learning of abductive knowledge bases for BALPs

In Chapter 3, we develop an approach to abductive plan recognition using Bayesian Logic Programs. We manually defined the structure of the BLP either based on planning knowledge or based on our knowledge of the domain. This approach is not always practical since it requires input from an expert. It would be desirable to develop a system that could automatically learn the knowledge base or the structure of the BLP from data. Such an approach would accept a set of observation literals and the top-level plan and learn a set of definite clauses that capture the causal structure between high level plans and observations. Existing approaches

to structure learning in BLPs (Kersting & De Raedt, 2008) typically learn deductive rules in which the observation literals entail the high level plan. Similarly, several existing inductive logic programming (ILP) based rule learning approaches (Quinlan, 1990; McCreath & Sharma, 1998; Srinivasan, 2001) also learn deductive rules. However, the task of abductive plan recognition requires abductive knowledge bases in which the causal structure between high level plans and observed actions is captured in the rules, i.e. the rule body should typically consist of a high level plan, while the head consists of an observed action or a subgoal. Existing work in learning abductive knowledge bases (Thompson & Mooney, 1994) mainly focuses on learning abductive propositional rules. A natural extension of our work on BALPs involves developing a novel approach to learning first-order abductive knowledge bases for tasks involving abductive reasoning.

6.1.2 Parameter learning from incomplete data for BLPs

6.1.2.1 Parameter learning using approximate inference techniques

In this dissertation, we have developed approaches using BLPs to solve two different tasks – abductive plan recognition (Chapter 3) and machine reading (Chapter 4 and Chapter 5). Both tasks share a common characteristic in that they involve learning from incomplete and partially observed data. For both the tasks, we have used the EM algorithm modified for BLPs (Kersting & De Raedt, 2008) to learn the parameters. While we were able to learn accurate parameters for the domains used in the plan recognition task, we were unsuccessful in learning useful parameters on the machine reading task, possibly due to the lack of sufficient data. We

also observed that EM was able to converge to local optima when exact inference was used to perform probabilistic inference in Bayesian networks. However, when approximate inference was used to perform probabilistic inference, the EM algorithm did not converge well. On most real domains, exact inference is not tractable. Extending existing approaches (Kulesza & Pereira, 2008; Wainwright, 2006) and developing new parameter learning approaches that can estimate parameters accurately when approximate inference is used is another direction for future work.

6.1.2.2 Discriminative learning of parameters

Most existing approaches to parameter learning in BLPs perform generative learning by optimizing the data log likelihood. However, a number of real world tasks including plan recognition would require parameters learned discriminatively by optimizing the conditional likelihood function. Several approaches (Greiner et al., 2002; Carvalho, Roos, Oliveira, & Myllymäki, 2011) to learning Bayesian network parameters discriminatively have been proposed. However, most of them assume that the training data is completely observed. A number of real world tasks like plan recognition involve incomplete data. Developing approaches that learn the BLP parameters discriminatively from incomplete or partially observed data is another topic for future work.

6.1.3 Lifted inference for BLPs and BALPs

Most real world tasks like plan recognition and machine reading involve large domains consisting of a large number of entities. Such domains result in the

construction of ground networks with several nodes, which eventually make probabilistic inference intractable. In the recent past, several approaches (Braz, Amir, & Roth, 2005; Milch, Zettlemoyer, Kersting, Haimes, & Kaelbling, 2008; Singla & Domingos, 2008; den Broeck, Taghipour, Meert, Davis, & De Raedt, 2011; Gogate & Domingos, 2011; Ahmadi, Kersting, & Natarajan, 2012) to lifted inference have been proposed. These lifted inference techniques are capable of performing probabilistic inference without having to construct ground networks. In other words, they perform probabilistic inference at the level of first-order clauses. However, most of these techniques have not yet been applied to real tasks. It would be useful to develop lifted inference techniques that can be applied to real tasks like plan recognition and machine reading for both BLPs and BALPs. Using lifted inference techniques will eliminate the need to construct ground networks thereby reducing time to inference. Further, they could also alleviate problems that arise from using approximate inference while learning the parameters for BLPs and BALPs.

6.2 Evaluation and task specific improvements

6.2.1 Plan Recognition

6.2.1.1 Other applications of BALPs

In Chapter 3, we developed an approach to abductive plan recognition using BLPs and evaluated the efficacy of our approach on three different domains – story understanding, strategic planning (Monroe), and intelligent user interfaces (Linux). Of these, Story Understanding and Linux data sets are real data sets, while Monroe was artificially generated. Our experimental evaluation demonstrated that it was

easier to achieve superior performance on the artificial domain than the real ones, possibly due to the presence of noise in the real data sets. Going forward, we would like to evaluate our BALP approach on other real world domains. One important real application of plan recognition lies in the area of monitoring activities of daily living for elderly care (Park & Kautz, 2008). In this domain, the plan recognition task involves monitoring the activities performed by an elderly person and detecting any abnormality in their behavior or activities performed so that an alarm could be raised at the right time. Typically, the activities performed can be recognized using an RFID reader that is capable of reading RFID tags that are attached to various objects in a smart home.

6.2.1.2 Improvements to BALPs for abductive plan recognition

We can extend our BALP approach to abductive plan recognition in several ways. In the current approach, we do not account for the order in which the observations are seen by the plan recognition system. For instance, in the Linux domain, the scenario in which a copy (cp) command is followed by a remove (rm) command is different from the one in which a remove (rm) command is followed by the copy (cp) command. Going forward, we could include temporal constraints to our current model to be able to account for the order observations. One obvious approach to incorporating temporal constraints involves using representation such as event calculus that can handle temporal constraints.

In our current approach, if there are multiple assumptions that can unify with a sub-goal, we use a greedy heuristic to match a sub-goal with existing assumption

literals. We perform this greedy matching to reduce the size of the ground network constructed. While this approach worked for the application domains considered in this dissertation, it would be interesting to explore domain specific heuristics for matching assumption literals with sub-goals in the future.

6.2.1.3 Comparison of BALPs to other SRL models

In this dissertation, we have compared the performance of BALPs to that of MLNs. We explored the possibility of comparing BALPs' performance to that of ProbLog (Kimmig et al., 2008), since ProbLog is known to subsume other probabilistic logics such as PRISM (Sato, 1995) and Poole's Horn Abduction (Poole, 1993). However, due to issues with the current implementation of ProbLog, we could not perform an experimental comparison of BALPs with ProbLog. It would be interesting to perform an experimental evaluation of BALPs comparing its performance to that of other probabilistic logics such as PRISM, Poole's Horn Abduction, and Abductive Stochastic Logic Programs (ASLPs) (Tamaddoni-Nezhad, Chaleil, Kakas, & Muggleton, 2006). ASLPs have been applied to abductive tasks in computational biology, but it would be interesting to apply ASLPs to our evaluation domains from the plan recognition task.

6.2.2 Machine Reading

6.2.2.1 Comparison of BLPs to other MLN approaches

In Chapter 4, we compare the performance of BLPs to a basic MLN based approach on the machine reading task. As mentioned earlier, several improvements

could be made to this basic MLN model to make it more competitive with BLPs. For instance, several spurious facts like employs(a,a) were inferred due to MLN's grounding process. These inferences can be prevented by including additional clauses in the MLN that impose integrity constraints that prevent such nonsensical propositions. Lack of strict typing on the arguments of relations in the IC ontology has also resulted in inferior performance of the MLNs. To overcome this, relations that do not have strictly defined types could be specialized. Lack of constrained or focused grounding is one of the reasons for poor performance of MLNs on the machine reading task. We could improve its performance by using approaches proposed to constrain the size of ground networks constructed by MLNs (Schoenmackers et al., 2008; Singla & Mooney, 2011). Further, techniques proposed by Sorower et al. (2011) can be incorporated to explicitly handle missing information in text in both BLPs and MLNs. We could also use techniques proposed by Niu et al. (2012) to scale MLNs to large corpora like natural language text.

6.2.2.2 Crowdsourcing for large scale evaluation

In Chapters 4 and 5, we manually evaluated additional facts inferred by the BLP framework on a small set of 40 documents from the IC data set to calculate precision. A natural extension of this work would be to perform large scale evaluation of inferred facts on a much larger test set. Additionally, we can explore the annotation of test documents with all possible facts that can be inferred for recall calculations as described in Chapter 4. Since the IC data set does not have ground truth information, manual evaluation on a larger test set is not practical. We

could employ crowdsourcing techniques such as Amazon Mechanical Turk (Snow, O'Connor, Jurafsky, & Ng, 2008) to perform a large scale evaluation.

We could also use the ground truth data collected from crowdsourcing to aid the EM algorithm to learn more accurate parameters for the first-order rules. In Chapters 4 and 5, we demonstrated that the amount of training data was not sufficient to learn accurate noisy-or parameters using EM on the machine reading task. Due to incomplete nature of natural language text, relations that are implicit seldom appear in training data. Further, due to the absence of ground truth information in the form of relations that can inferred from explicitly stated facts, the training data is always incomplete. Using a larger training set might solve the problem to some extent. A more reasonable solution would be to provide additional supervision to the learning algorithm in the form of ground truth information, i.e. relations that can be inferred from explicitly stated facts. We hypothesize that weak supervision in the form of noisy ground truth information collected from crowdsourcing could still help improve the performance of the EM algorithm for learning noisy-or parameters.

Zeichner et al.(2012) have proposed a framework to evaluate textual entailment rules using crowdsourcing techniques. Textual entailment rules are similar to the first-order rules learned from natural language text in this dissertation, but are much simpler with the antecedent consisting of a single literal. We could explore the possibility of applying such techniques to evaluate the first-order rules learned in this dissertation as well.

6.2.2.3 Alternate approaches to scoring rules using lexical semantics

In Chapter 5, we proposed a novel approach to scoring first-order rules using WordNet for machine reading. A logical extension of our work involves experimenting with different similarity metrics developed for WordNet. Since a number of existing similarity metrics do not compute a scaled score between 0 and 1, we did not explore using these similarity metrics in our work. However, it would be interesting to explore alternate similarity metrics and approaches to scaling to compute a final score between 0 and 1. Further, we could also develop a novel similarity metric that is better suited for scoring first-order rules for the machine reading task. Exploring alternate approaches such as distributional lexical semantics (Garrette et al., 2011) to compute word similarity for scoring rules is another topic for future work. Unlike using a curated ontology such as WordNet to calculate similarity scores between words, approaches from distributional lexical semantics calculate similarity measures based on the frequency of occurrence of word pairs in the same context. Using weights computed by these approaches as initial weights for parameter learning might help avoid local maxima and converge to a better set of parameters. Finally, it would be interesting to compare the weights calculated by these approaches with human judgments as well.

6.2.2.4 Evaluation of multi-relational online rule learning using larger corpora

In Chapter 5, we evaluated our online rule learner on the IC data set for machine reading. Even though the rule learner is capable of learning rules with

multiple relation literals in the antecedent, we restricted learning rules with a single relation in the antecedent due to lack of sufficient data. In future, we would like to evaluate the online rule learner using a much larger corpora so that we can learn more useful rules consisting of multiple relations in the antecedent.

6.2.2.5 Improving the recall of IE system

We could use the BLP approach described in Chapter 4 to improve the recall of the underlying IE system. If the IE system fails to extract a fact that is explicitly stated and if this fact is inferred by the BLP approach, then augmenting the output of the IE system with the inferred facts will result in the improvement of the overall recall of the IE system. Nahm and Mooney (2000) demonstrated the efficacy of this approach to improve the recall of an IE system on a computer-related job-posting domain. We could also use the inferred facts as additional training data to learn new rules and parameters. Since marginal probabilities are available for inferred facts, these inferences could be used as "virtual evidence" (Pearl, 1988) during probabilistic inference and learning. Most IE systems output relation instances that have high confidence scores. It would be interesting to use the inferences made by the BLP approach to improve the evidence for those extractions that have lower confidence scores, thereby improving the recall of the underlying system further. A similar approach has been taken by Welty et al. (2012) to improve the recall on a question answering task.

Chapter 7

Conclusion

This dissertation focuses on developing approaches using Bayesian Logic Programs (BLPs) to solve two real world tasks – plan recognition and machine reading. Plan recognition involves inferring an intelligent agent's top-level plans based on its observed actions. It has practical applications in several domains including monitoring activities of daily living for elderly care, intelligent surveillance systems, and intelligent personal assistants. Machine reading involves automatic extraction of information from natural language text. Like plan recognition, machine reading is also widely used in practical applications such as deep question answering. For these reasons, we have focused on developing approaches using BLPs on these two tasks. Further, both tasks share a common characteristic in that they involve learning and inference from incomplete or partially observed data. In this dissertation, we demonstrate the efficacy of BLPs for inference and learning from partially observed data.

One of the main advantages of BLPs over other SRL models such as MLNs lies in its ability to perform focused grounding by constructing ground Bayesian networks consisting of only those literals that are used to prove the query. We chose to use BLPs in this dissertation as we hypothesized that this aspect of BLPs

would help solve real world tasks involving large domains. Further, we also hypothesized that the directed nature of BLPs could provide the additional flexibility to use different types of logical inference, thereby enabling us to use BLPs on a variety of real world tasks. Due to these aspects of BLPs, we hypothesized that they are better suited for solving several tasks when compared to their undirected counterparts, MLNs. In this dissertation, we demonstrate empirically that our hypothesis is valid by demonstrating the superior performance of BLPs over MLNs on both plan recognition and machine reading.

In the first part of the dissertation, we developed an approach to abductive plan recognition using BLPs. Since BLPs use SLD resolution for logical inference, they cannot be used for abductive reasoning tasks like plan recognition. Here, our primary contribution involves extending BLPs for abductive reasoning by using logical abduction instead of deduction to construct ground Bayesian networks. In logical abduction, missing facts are assumed when necessary to complete proof trees, and we use the resulting abductive proof trees to construct Bayesian networks. We call the resulting model Bayesian Abductive Logic Programs (BALPs). We learned the parameters for the BALP framework automatically from data using the Expectation Maximization algorithm adapted for BLPs by Kersting and De Raedt (2008). Experimental evaluation on three benchmark data sets demonstrated that BALPs outperform the existing state-of-art methods such as MLNs for plan recognition. Here, we demonstrated that the superior performance of BLPs over MLNs is not only due to its ability to perform constrained grounding, but also due to its directed nature, which allows to easily use logical abduction instead of standard

logical deduction to construct ground Bayesian networks.

In the second part of the dissertation, we developed approaches to machine reading using BLPs. We first developed an approach using BLPs to infer implicit facts from natural language text. Our approach involves learning probabilistic rules in first-order logic from a large corpus of extracted facts using LIME (McCreath & Sharma, 1998), an existing rule learner and then using the resulting BLP to make effective probabilistic inferences when interpreting new documents. Experimental evaluation of our system on a realistic test corpus from DARPA's Machine Reading project demonstrated improved performance compared to a purely logical approach based on ILP, and an alternative approach based on MLNs. Here again, we demonstrated empirically that the superior performance of BLPs over MLNs is due to its focused grounding that resulted in the construction of smaller ground networks when compared to those constructed by MLNs, which eventually allows BLPs to scale to large corpora of natural language text.

Next, we developed a novel online rule learner that is capable of learning first-order rules from noisy and incomplete natural language extractions, which are then used to infer implicit facts from natural language text. The rule learner handles the concise, incomplete nature of natural-language text by learning rules in which the body of the rule typically consists of relations that are frequently explicitly stated, while the head is a relation that is more typically inferred. We use the frequency of occurrence of extracted relations as a heuristic for distinguishing those that are typically explicitly stated from the ones that are usually inferred. In order to allow scaling to large corpora, we developed an efficient online rule learner. Exper-

imental evaluation on the machine reading task demonstrated superior performance of our rule learner when compared to LIME, an existing rule learner used in our previous approach.

Finally, we developed a novel approach to scoring first-order rules learned from IE extractions for the purpose of inferring implicit facts from natural language text for machine reading. Our approach uses the lexical information from a curated ontology such as WordNet (Fellbaum, 1998) to score the rules. The basic idea behind our approach is that more accurate rules typically have predicates that are closely related to each other in terms of the meanings of the English words used to name them. Since WordNet is a rich resource for lexical information, we use it for scoring rules based on word similarity. Experimental evaluation on the machine reading task demonstrated superior performance of the our approach over both manual weights and weights learned using EM.

Overall, this dissertation makes important contributions towards learning and inference from incomplete data using Bayesian Logic Programs. We demonstrate the efficacy of BLPs on two diverse real-world tasks – plan recognition and machine reading, both of which have a wide variety of applications in several domains. The approaches developed in this dissertation make advances in the area of performing inference by integrating evidence from several different sources. Our approaches to machine reading have the potential to enable computers to read and understand language better. Further, we demonstrate the superior performance of BLPs over their undirected counterparts, MLNs on both the tasks. The approaches developed in this dissertation also have a direct impact on the advancement of com-

mercial applications that use plan recognition and machine reading. Today, plan recognition is already being used in commercial intelligent personal assistant systems such as Siri. Our own work in plan recognition is motivated by the application in elderly care for monitoring activities of daily living (Park & Kautz, 2008). Similarly, our contributions in machine reading can be directly used in intelligent personal assistants such as Siri for deep question answering. Further, these topics are of immediate interest to defense and security agencies for quick processing of huge amounts of natural language text for automatic question answering. In a nutshell, we firmly believe that this dissertation makes several contributions towards advancement of commercial applications as well as scientific research.

Appendices

Appendix A

Details of Plan Recognition Datasets

This appendix gives additional details about the data sets used to evaluate BALPs on the plan recognition task.

A.1 Monroe

The knowledge base constructed from planning knowledge for the Monroe domain is given below. For additional details on the data set creation, we refer the reader to Nate Blaylock's Ph.D. dissertation (2005).

```
# set-up-shelter sets up a shelter at a certain location
get-electricity(Loc)|set-up-shelter(Loc).
person-get-to(Leader,Loc)|set-up-shelter(Loc),shelter-leader(Leader).
object-get-to(Foodx,Loc)|set-up-shelter(Loc),food(Foodx).

#fix-water-main
shut-off-water(From,To)|fix-water-main(From,To).
repair-pipe(From,To)|fix-water-main(From,To).
turn-on-water(From,To)|fix-water-main(From,To).

# clear-road-hazard  cleans up a hazardous spill
block-road(From,To)|clear-road-hazard(From,To).
clean-up-hazard(From,To)|clear-road-hazard(From,To).
unblock-road(From,To)|clear-road-hazard(From,To).

# clear-road-wreck  gets a wreck out of the road
set-up-cones(From,To)|clear-road-wreck(From,To).
clear-wreck(From,To)|clear-road-wreck(From,To).
take-down-cones(From,To)|clear-road-wreck(From,To).

# clear-road-tree
set-up-cones(From,To)|clear-road-tree(From,To).
clear-tree(Treex)|clear-road-tree(From,To),tree-blocking-road(From,To,Treex).
take-down-cones(From,To)|clear-road-tree(From,To).
```

```
# plow-road
navegate-snowplow(Driver,Plow,From)|plow-road(From,To),
plowdriver(Driver),snowplow(Plow).
engage-plow(Driver,Plow)|plow-road(From,To),plowdriver(Driver),
snowplow(Plow).
navegate-snowplow(Driver,Plow,To)|plow-road(From,To),plowdriver(Driver),
snowplow(Plow).
disengage-plow(Driver,Plow)|plow-road(From,To),plowdriver(Driver),
snowplow(Plow).

#quell-riot
declare-curfew(Townx)|quell-riot(Loc),in-town(Loc,Townx).
person-get-to(Px,Loc)|quell-riot(Loc),police-unit(Px),in-town(Loc,Townx).
set-up-barricades(Px)|quell-riot(Loc),police-unit(Px),in-town(Loc,Townx).

#provide-temp-heat
person-get-to(Person1,Ploc)|provide-temp-heat(Person1),shelter(Ploc),
person(Person1).
generate-temp-electricity(Ploc)|provide-temp-heat(Personx),person(Personx),
atloc(Personx,Ploc).
turn-on-heat(Ploc)|provide-temp-heat(Personx),person(Personx),
atloc(Personx,Ploc).

#fix-power-line
crew-get-to(Crew,Lineloc)|fix-power-line(Lineloc),power-crew(Crew).
vehicle-get-to(Van,Lineloc)|fix-power-line(Lineloc),power-van(Van).
repair-line(Crew,Lineloc)|fix-power-line(Lineloc),power-crew(Crew).

#provide-medical-attention
person-get-to(Personx,Hosp)|provide-medical-attention(Personx),
hospital(Hosp).
treat-in-hospital(Personx,Hosp)|provide-medical-attention(Personx),
hospital(Hosp).
emt-treat(Personx)|provide-medical-attention(Personx).

# clean-up-hazard
call(fema)|clean-up-hazard(From,To),hazard-seriousness(From,To,very-hazardous).
crew-get-to(Ht,From)|clean-up-hazard(From,To),hazard-team(Ht).
clean-hazard(Ht,From,To)|clean-up-hazard(From,To),hazard-team(Ht).

# block-road - blocks off a road
set-up-cones(From,To)|block-road(From,To).
person-get-to(Police,From)|block-road(From,To),police-unit(Police).

# unblock-road - unblocks a road
take-down-cones(From,To)|unblock-road(From,To).

# get-electricity provides electricity to a site (if not already there)
generate-temp-electricity(Loc)|get-electricity(Loc),no-electricity(Loc).

# repair-pipe
crew-get-to(Crew,From)|repair-pipe(From,To),water-crew(Crew).
set-up-cones(From,To)|repair-pipe(From,To).
open-hole(From,To)|repair-pipe(From,To).
replace-pipe(Crew,From,To)|repair-pipe(From,To),water-crew(Crew).
```

```
close-hole(From,To)|repair-pipe(From,To).
take-down-cones(From,To)|repair-pipe(From,To).

# open-hole
vehicle-get-to(Backhoex,From)|open-hole(From,To),backhoe(Backhoex).
dig(Backhoex,From)|open-hole(From,To),backhoe(Backhoex).

#close-hole
vehicle-get-to(Backhoex,From)|close-hole(From,To),backhoe(Backhoex).
fill-in(Backhoex,From)|close-hole(From,To),backhoe(Backhoex).

# set-up-cones
crew-get-to(Crew,From)|set-up-cones(From,To),work-crew(Crew).
place-cones(Crew)|set-up-cones(From,To),work-crew(Crew).

# take-down-cones
crew-get-to(Crew,From)|take-down-cones(From,To),work-crew(Crew).
pickup-cones(Crew)|take-down-cones(From,To),work-crew(Crew).

# clear-wreck
tow-to(Veh,Dumpx)|clear-wreck(From,To),
wrecked-vehicle(From,To,Veh),garbage-dump(Dumpx).

# tow-to - tows a vehicle somewhere
vehicle-get-to(Ttruck,Vehloc)|tow-to(Veh,To),tow-truck(Ttruck),
vehicle(Veh),atloc(Veh,Vehloc).
hook-to-tow-truck(Ttruck,Veh)|tow-to(Veh,To),tow-truck(Ttruck).
vehicle-get-to(Ttruck,To)|tow-to(Veh,To),tow-truck(Ttruck).
unhook-from-tow-truck(Ttruck,Veh)|tow-to(Veh,To),tow-truck(Ttruck).

# clear-tree
crew-get-to(Tcrew,Treeloc)|clear-tree(Treex),tree-crew(Tcrew),
tree(Treex),atloc(Treex,Treeloc).
cut-tree(Tcrew,Treex)|clear-tree(Treex),tree-crew(Tcrew).
remove-blockage(Treex)|clear-tree(Treex).

# remove-blockage
crew-get-to(Crew,Loc)|remove-blockage(Stuff),work-crew(Crew),atloc(Stuff,Loc).
carry-blockage-out-of-way(Crew,Stuff)|remove-blockage(Stuff),work-crew(Crew).
object-get-to(Stuff,Dumpx)|remove-blockage(Stuff),garbage-dump(Dumpx).

# declare-curfew
call(ebs)|declare-curfew(Townx).
call(police-chief)|declare-curfew(Townx).

# generate-temp-electricity
make-full-fuel(Gen)|generate-temp-electricity(Loc),generator(Gen).
object-get-to(Gen,Loc)|generate-temp-electricity(Loc),generator(Gen).
hook-up(Gen,Loc)|generate-temp-electricity(Loc),generator(Gen).
turn-on(Gen)|generate-temp-electricity(Loc),generator(Gen).

# make-full-fuel - makes sure arg1 is full of fuel
object-get-to(Gc,Ss)|make-full-fuel(Gen),gas-can(Gc),service-station(Ss).
add-fuel(Ss,Gc)|make-full-fuel(Gen),gas-can(Gc),service-station(Ss).
object-get-to(Gc,Genloc)|make-full-fuel(Gen),gas-can(Gc),atloc(Gen,Genloc).
pour-into(Gc,Gen)|make-full-fuel(Gen),gas-can(Gc).
```

```
object-get-to(Gen,Ss)|make-full-fuel(Gen),service-station(Ss).
add-fuel(Ss,Gen)|make-full-fuel(Gen),service-station(Ss).

# add-fuel (at service-station)
pay(Ss)|add-fuel(Ss,Obj).
pump-gas-into(Ss,Obj)|add-fuel(Ss,Obj).

# repair-line
shut-off-power(Crew,Lineloc)|repair-line(Crew,Lineloc).
clear-tree(Treex)|repair-line(Crew,Lineloc),tree(Treex).
remove-wire(Crew,Lineloc)|repair-line(Crew,Lineloc).
string-wire(Crew,Lineloc)|repair-line(Crew,Lineloc).
turn-on-power(Crew,Lineloc)|repair-line(Crew,Lineloc).

# shut-off-power
call(Powerco)|shut-off-power(Crewx,Loc),in-town(Loc,Town1),
powerco-of(Town1,Powerco),power-crew(Crewx).

# turn-on-power
call(Powerco)|turn-on-power(Crewx,Loc),in-town(Loc,Townx),
powerco-of(Townx,Powerco),power-crew(Crewx).

# shut-off-water
call(Waterco)|shut-off-water(From,To),in-town(From,Townx),
waterco-of(Townx,Waterco).

# turn-on-water
call(Waterco)|turn-on-water(From,To),in-town(From,Townx),
waterco-of(Townx,Waterco).

# emt-treat
crew-get-to(Emt,Personloc)|emt-treat(Personx),emt-crew(Emt),
atloc(Personx,Personloc).
treat(Emt,Personx)|emt-treat(Personx),emt-crew(Emt).

# stabilize
emt-treat(Personx)|stabilize(Personx).

# get-to
drive-to(Personx,Veh,Place)|person-get-to(Personx,Place),person(Personx),
vehicle(Veh).
drive-to(Crewx,Veh,Place)|crew-get-to(Crewx,Place),work-crew(Crewx),
vehicle(Veh).
drive-to(Personx,Veh,Place)|vehicle-get-to(Veh,Place),person(Personx),
vehicle(Veh).
vehicle-get-to(Veh,Objplace)|object-get-to(Obj,Place),atloc(Obj,Objplace),
vehicle(Veh).
get-in(Obj,Veh)|object-get-to(Obj,Place),vehicle(Veh).
vehicle-get-to(Veh,Place)|object-get-to(Obj,Place),vehicle(Veh).
get-out(Obj,Veh)|object-get-to(Obj,Place),vehicle(Veh).
navegate-vehicle(Personx,Veh,Loc)|drive-to(Personx,Veh,Loc).
load(Personx,Obj,Veh)|get-in(Obj,Veh),person(Personx).
unload(Personx,Obj,Veh)|get-out(Obj,Veh),person(Personx).
climb-in(Personx,Veh)|get-in(Personx,Veh).
climb-out(Personx,Veh)|get-out(Personx,Veh).
```

```
# axioms

# points
point(X)|hospital(X).
point(X)|garbage-dump(X).
point(X)|park(X).
point(X)|shelter(X).
point(X)|school(X).
point(X)|university(X).
point(X)|mall(X).
point(X)|transport-hub(X).
point(X)|service-station(X).

# vehicles
vehicle(X)|ambulance(X).
vehicle(X)|bus(X).
vehicle(X)|power-van(X).
vehicle(X)|truck(X).
vehicle(X)|police-van(X).
vehicle(X)|car(X).
vehicle(X)|wrecked-car(X).
truck(X)|dump-truck(X).
truck(X)|tow-truck(X).
truck(X)|water-truck(X).
truck(X)|backhoe(X).
truck(X)|snowplow(X).
fit-in(Obj,Veh)|vehicle(Veh),person(Obj).
fit-in(Obj,Veh)|dump-truck(Veh),tree(Obj).
fit-in(Obj,Veh)|vehicle(Veh),generator(Obj).
fit-in(Obj,Veh)|vehicle(Veh),food(Obj).
can-drive(Personx,Veh)|ambulance(Veh),emt-crew(Personx).
can-drive(Personx,Veh)|power-van(Veh),power-crew(Personx).
can-drive(Personx,Veh)|truck(Veh),truck-driver(Personx).
can-drive(Personx,Veh)|bus(Veh),bus-driver(Personx).
can-drive(Personx,Veh)|police-van(Veh),police-unit(Personx).
can-drive(Personx,Veh)|tow-truck(Veh),tow-truck-driver(Personx).
can-drive(Personx,Veh)|backhoe(Veh),construction-crew(Personx).
can-drive(Personx,Veh)|water-truck(Veh),water-crew(Personx).
can-drive(Personx,Veh)|snowplow(Veh),plowdriver(Personx).
can-drive(Personx,Veh)|adult(Personx).

# people
person(X)|adult(X).
person(X)|child(X).
adult(X)|emt-crew(X).
adult(X)|work-crew(X).
adult(X)|bus-driver(X).
adult(X)|truck-driver(X).
adult(X)|police-unit(X).
adult(X)|tow-truck-driver(X).
adult(X)|plowdriver(X).
adult(X)|shelter-leader(X).

# lifting
can-lift(Personx,Obj)|person(Obj),emt-crew(Personx).
```

```
# work-crew
work-crew(X)|power-crew(X).
work-crew(X)|tree-crew(X).
work-crew(X)|construction-crew(X).
work-crew(X)|water-crew(X).
work-crew(X)|hazard-team(X).

# roads and locs
atloc(Obj,Loc)|on-road(Obj,Loc,To).
```

A.2 Reformulated Monroe

The knowledge based constructed for Reformulated Monroe is given below:

```
# set-up-shelter sets up a shelter at a certain location
get-electricity(Locx)|set-up-shelter(Locx).
person-get-to(Leaderx,Locx)|set-up-shelter(Locx),shelter-leader(Leaderx).
object-get-to(Foodx,Locx)|set-up-shelter(Locx),food(Foodx).

#fix-water-main
shut-off-water(From,To)|fix-water-main(From,To).
repair-pipe(From,To)|fix-water-main(From,To).
turn-on-water(From,To)|fix-water-main(From,To).

# clear-road-hazard  cleans up a hazardous spill
block-road(From,To)|clear-road-hazard(From,To).
clean-up-hazard(From,To)|clear-road-hazard(From,To).
unblock-road(From,To)|clear-road-hazard(From,To).

# clear-road-wreck  gets a wreck out of the road
set-up-cones(From,To)|clear-road-wreck(From,To).
clear-wreck(From,To)|clear-road-wreck(From,To).
take-down-cones(From,To)|clear-road-wreck(From,To).

# clear-road-tree
set-up-cones(From,To)|clear-road-tree(From,To).
clear-tree(Treex,From,To)|clear-road-tree(From,To),
tree-blocking-road(From,To,Treex).
take-down-cones(From,To)|clear-road-tree(From,To).

# plow-road
engage-plow(Driver,Plow)|plow-road(From,To),plowdriver(Driver),
snowplow(Plow).
navegate-snowplow(Driver,Plow,From,To)|plow-road(From,To),
plowdriver(Driver),snowplow(Plow).
disengage-plow(Driver,Plow)|plow-road(From,To),plowdriver(Driver),
snowplow(Plow).

#quell-riot
declare-curfew(Townx)|quell-riot(Locx),in-town(Locx,Townx).
person-get-to(Px,Locx)|quell-riot(Locx),police-unit(Px),in-town(Locx,Townx).
set-up-barricades(Px)|quell-riot(Locx),police-unit(Px),in-town(Locx,Townx).
```

```
#provide-temp-heat
person-get-to(Person1,Ploc)|provide-temp-heat(Person1),shelter(Ploc),
person(Person1).
generate-temp-electricity(Ploc)|provide-temp-heat(Person1),person(Person1),
atloc(Person1,Ploc).
turn-on-heat(Ploc,Person1)|provide-temp-heat(Person1),person(Person1),
atloc(Person1,Ploc).

#fix-power-line
crew-get-to(Crew,Lineloc)|fix-power-line(Lineloc),power-crew(Crew).
vehicle-get-to(Van,Lineloc)|fix-power-line(Lineloc),power-van(Van).
repair-line(Crew,Lineloc)|fix-power-line(Lineloc),power-crew(Crew).

#provide-medical-attention
person-get-to(Personx,Hosp)|provide-medical-attention(Personx),
hospital(Hosp).
treat-in-hospital(Personx,Hosp)|provide-medical-attention(Personx),
hospital(Hosp).
emt-treat(Personx)|provide-medical-attention(Personx).

# clean-up-hazard
call(fema,From)|clean-up-hazard(From,To),
hazard-seriousness(From,To,very-hazardous).
clear(From,To)|clean-up-hazard(From,To).
crew-get-to(Ht,From)|clean-up-hazard(From,To),hazard-team(Ht).
clean-hazard(Ht,From,To)|clean-up-hazard(From,To),hazard-team(Ht).

# block-road - blocks off a road
set-up-cones(From,To)|block-road(From,To).
person-get-to(Policex,From)|block-road(From,To),police-unit(Policex).

# unblock-road - unblocks a road
take-down-cones(From,To)|unblock-road(From,To).

# get-electricity provides electricity to a site (if not already there)
generate-temp-electricity(Locx)|get-electricity(Locx),no-electricity(Locx).

# repair-pipe
crew-get-to(Crewx,From)|repair-pipe(From,To),water-crew(Crewx).
set-up-cones(From,To)|repair-pipe(From,To).
open-hole(From,To)|repair-pipe(From,To).
replace-pipe(Crewx,From,To)|repair-pipe(From,To),water-crew(Crewx).
close-hole(From,To)|repair-pipe(From,To).
take-down-cones(From,To)|repair-pipe(From,To).

# open-hole
vehicle-get-to(Backhoex,From)|open-hole(From,To),backhoe(Backhoex).
dig(Backhoex,From,To)|open-hole(From,To),backhoe(Backhoex).

#close-hole
vehicle-get-to(Backhoex,From)|close-hole(From,To),backhoe(Backhoex).
fill-in(Backhoex,From,To)|close-hole(From,To),backhoe(Backhoex).

# set-up-cones
crew-get-to(Crewx,From)|set-up-cones(From,To),work-crew(Crewx).
```

```
place-cones(Crewx,From,To)|set-up-cones(From,To),work-crew(Crewx).

# take-down-cones
crew-get-to(Crewx,From)|take-down-cones(From,To),work-crew(Crewx).
pickup-cones(Crewx,From,To)|take-down-cones(From,To),work-crew(Crewx).

# clear-wreck
tow-to(Vehx,Dumpx)|clear-wreck(From,To),wrecked-vehicle(From,To,Vehx),
garbage-dump(Dumpx).
clear(From,To)|clear-wreck(From,To).

# tow-to - tows a vehicle somewhere
vehicle-get-to(Ttruck,Vehloc)|tow-to(Veh,To),tow-truck(Ttruck),vehicle(Veh),
atloc(Veh,Vehloc).
hook-to-tow-truck(Ttruck,Veh)|tow-to(Veh,To),tow-truck(Ttruck).
vehicle-get-to(Ttruck,To)|tow-to(Veh,To),tow-truck(Ttruck).
unhook-from-tow-truck(Ttruck,Veh)|tow-to(Veh,To),tow-truck(Ttruck).

# clear-tree
crew-get-to(Tcrew,Treeloc)|clear-tree(Treex,From,To),tree-crew(Tcrew),
tree(Treex),atloc(Treex,Treeloc).
cut-tree(Tcrew,Treex,From,To)|clear-tree(Treex,From,To),tree-crew(Tcrew).
remove-blockage(Treex,From,To)|clear-tree(Treex,From,To).

# remove-blockage
crew-get-to(Crewx,Loc)|remove-blockage(Stuff,From,To),work-crew(Crewx),
atloc(Stuff,Loc).
carry-blockage-out-of-way(Crewx,Stuff,From,To)|remove-blockage(Stuff,From,To),
work-crew(Crewx).
object-get-to(Stuffx,Dumpx)|remove-blockage(Stuffx,From,To),garbage-dump(Dumpx).

# declare-curfew
call(ebs,Townx)|declare-curfew(Townx).
call(police-chief,Townx)|declare-curfew(Townx).

# generate-temp-electricity
make-full-fuel(Genx)|generate-temp-electricity(Locx),generator(Genx).
object-get-to(Genx,Locx)|generate-temp-electricity(Locx),generator(Genx).
hook-up(Genx,Locx)|generate-temp-electricity(Locx),generator(Genx).
turn-on(Genx)|generate-temp-electricity(Locx),generator(Genx).

# make-full-fuel - makes sure arg1 is full of fuel
object-get-to(Gc,Ss)|make-full-fuel(Gen),gas-can(Gc),service-station(Ss).
add-fuel(Ss,Gc)|make-full-fuel(Gen),gas-can(Gc),service-station(Ss).
object-get-to(Gc,Genloc)|make-full-fuel(Gen),gas-can(Gc),atloc(Gen,Genloc).
pour-into(Gc,Gen)|make-full-fuel(Gen),gas-can(Gc).
object-get-to(Gen,Ss)|make-full-fuel(Gen),service-station(Ss).
add-fuel(Ss,Gen)|make-full-fuel(Gen),service-station(Ss).

# add-fuel (at service-station)
pay(Ss)|add-fuel(Ss,Obj).
pump-gas-into(Ss,Obj)|add-fuel(Ss,Obj).

# repair-line
shut-off-power(Crewx,Lineloc)|repair-line(Crewx,Lineloc).
clear-tree(Treex,From,To)|repair-line(Crewx,Lineloc),tree(Treex).
```

```
remove-wire(Crewx,Lineloc)|repair-line(Crewx,Lineloc).
string-wire(Crewx,Lineloc)|repair-line(Crewx,Lineloc).
turn-on-power(Crewx,Lineloc)|repair-line(Crewx,Lineloc).

# shut-off-power
call(Powerco,Loc)|shut-off-power(Crewx,Loc),in-town(Loc,Townx),
powerco-of(Townx,Powerco),power-crew(Crewx).

# turn-on-power
call(Powerco,Locx)|turn-on-power(Crewx,Locx),in-town(Loc,Town1),
powerco-of(Town1,Powerco),power-crew(Crewx).

# shut-off-water
call(Waterco,From)|shut-off-water(From,To),in-town(From,Town1),
waterco-of(Town1,Waterco).
water-off(From,To)|shut-off-water(From,To).

# turn-on-water
call(Waterco,From)|turn-on-water(From,To),in-town(From,Townx),
waterco-of(Townx,Waterco).
water-on(From,To)|turn-on-water(From,To).

# emt-treat
crew-get-to(Emt,Personloc)|emt-treat(Personx),emt-crew(Emt),
atloc(Personx,Personloc).
treat(Emt,Personx)|emt-treat(Personx),emt-crew(Emt).

# stabilize
emt-treat(Personx)|stabilize(Personx).

# get-to
drive-to(Personx,Veh,Place)|person-get-to(Personx,Place),person(Personx),
vehicle(Veh).
drive-to(Crewx,Veh,Place)|crew-get-to(Crewx,Place),work-crew(Crewx),
vehicle(Veh).
drive-to(Personx,Veh,Place)|vehicle-get-to(Veh,Place),person(Personx),
vehicle(Veh).
vehicle-get-to(Veh,Objplace)|object-get-to(Obj,Place),atloc(Obj,Objplace),
vehicle(Veh).
get-in(Obj,Veh)|object-get-to(Obj,Place),vehicle(Veh).
vehicle-get-to(Veh,Place)|object-get-to(Obj,Place),vehicle(Veh).
get-out(Obj,Veh)|object-get-to(Obj,Place),vehicle(Veh).
navegate-vehicle(Person1,Veh,Loc)|drive-to(Person1,Veh,Loc).
load(Person1,Obj,Veh)|get-in(Obj,Veh),person(Person1).
unload(Person1,Obj,Veh)|get-out(Obj,Veh),person(Person1).
climb-in(Person1,Veh)|get-in(Person1,Veh).
climb-out(Person1,Veh)|get-out(Person1,Veh).

# axioms

# points
point(X)|hospital(X).
point(X)|garbage-dump(X).
point(X)|park(X).
point(X)|shelter(X).
point(X)|school(X).
```

```
point(X)|university(X).
point(X)|mall(X).
point(X)|transport-hub(X).
point(X)|service-station(X).

# vehicles
vehicle(X)|ambulance(X).
vehicle(X)|bus(X).
vehicle(X)|power-van(X).
vehicle(X)|truck(X).
vehicle(X)|police-van(X).
vehicle(X)|car(X).
vehicle(X)|wrecked-car(X).
truck(X)|dump-truck(X).
truck(X)|tow-truck(X).
truck(X)|water-truck(X).
truck(X)|backhoe(X).
truck(X)|snowplow(X).
fit-in(Obj,Veh)|vehicle(Veh),person(Obj).
fit-in(Obj,Veh)|dump-truck(Veh),tree(Obj).
fit-in(Obj,Veh)|vehicle(Veh),generator(Obj).
fit-in(Obj,Veh)|vehicle(Veh),food(Obj).
can-drive(Person1,Veh)|ambulance(Veh),emt-crew(Person1).
can-drive(Person1,Veh)|power-van(Veh),power-crew(Person1).
can-drive(Person1,Veh)|truck(Veh),truck-driver(Person1).
can-drive(Person1,Veh)|bus(Veh),bus-driver(Person1).
can-drive(Person1,Veh)|police-van(Veh),police-unit(Person1).
can-drive(Person1,Veh)|tow-truck(Veh),tow-truck-driver(Person1).
can-drive(Person1,Veh)|backhoe(Veh),construction-crew(Person1).
can-drive(Person1,Veh)|water-truck(Veh),water-crew(Person1).
can-drive(Person1,Veh)|snowplow(Veh),plowdriver(Person1).
can-drive(Person1,Veh)|adult(Person1).

# people
person(X)|adult(X).
person(X)|child(X).
adult(X)|emt-crew(X).
adult(X)|work-crew(X).
adult(X)|bus-driver(X).
adult(X)|truck-driver(X).
adult(X)|police-unit(X).
adult(X)|tow-truck-driver(X).
adult(X)|plowdriver(X).
adult(X)|shelter-leader(X).

# lifting
can-lift(Person1,Obj)|person(Obj),emt-crew(Person1).

# work-crew
work-crew(X)|power-crew(X).
work-crew(X)|tree-crew(X).
work-crew(X)|construction-crew(X).
work-crew(X)|water-crew(X).
work-crew(X)|hazard-team(X).

# roads and locs
```

```
atloc(Obj,Loc)|on-road(Obj,Loc,To).
```

A.3 Linux

The knowledge base constructed for the Linux domain is given below. For additional details on the data set creation, we refer the reader to Nate Blaylock's Ph.D. dissertation (2005).

```
#find a file named arg1
find(Prename,Name,Size,Prepath,Path)|
find-file-by-attr-name-exact(Name),file-prepath(Prename),
file-size(Size),source-dir-prepath(Prepath),source-dir-name(Path).
cd(Prepath,Path)|
find-file-by-attr-name-exact(Name),source-dir-prepath(Prepath),
source-dir-name(Path).
ls(Path,Name)|
find-file-by-attr-name-exact(Name),source-dir-prepath(Path),
file-name(Name).

#find a file that ends in arg1
find(Prename,Ext,Size,Prepath,Path)|
find-file-by-attr-name-ext(Ext),file-prepath(Prename),file-size(Size),
source-dir-prepath(Prepath),source-dir-name(Path).
cd(Prepath,Path)|
find-file-by-attr-name-ext(Ext),source-dir-prepath(Prepath),
source-dir-name(Path).
ls(Path,Ext)|
find-file-by-attr-name-ext(Ext),source-dir-prepath(Path),file-name(Ext).

#find a file that begins with arg1
find(Prename,Stem,Size,Prepath,Path)|
find-file-by-attr-name-stem(Stem),file-prepath(Prename),
file-size(Size),source-dir-prepath(Prepath),source-dir-name(Path).
cd(Prepath,Path)|
find-file-by-attr-name-stem(Stem),source-dir-prepath(Prepath),
source-dir-name(Path).
ls(Path,Stem)|
find-file-by-attr-name-stem(Stem),source-dir-prepath(Path),file-name(Stem).

#find a file that was last modified arg1
cd(Prepath,Path)|
find-file-by-attr-date-modification-exact(Date),source-dir-prepath(Prepath),
source-dir-name(Path).
ls(Path,Name)|
find-file-by-attr-date-modification-exact(Date),source-dir-prepath(Path),
file-name(Name).
egrep(Date,Prepath,Path)|
find-file-by-attr-date-modification-exact(Date),file-prepath(Prepath),
file-name(Path).
```

```
fgrep(Date,Prepath,Path)|
find-file-by-attr-date-modification-exact(Date),file-prepath(Prepath),
file-name(Path).
grep(Date,Prepath,Path)|
find-file-by-attr-date-modification-exact(Date),file-prepath(Prepath),
file-name(Path).

#compress all directories named arg1
compress(Prepath,Dirname)|
compress-dirs-by-attr-name(Dirname),source-dir-prepath(Prepath),
source-dir-name(Dirname).
gtar(Destprepath,Destpath,Sourceprepath,Dirname)|
compress-dirs-by-attr-name(Dirname),file-prepath(Destprepath),
file-name(Destpath),source-dir-prepath(Sourceprepath),
source-dir-name(Dirname).
gzip(Prepath,Dirname)|
compress-dirs-by-attr-name(Dirname),source-dir-prepath(Prepath),
source-dir-name(Dirname).
tar(Destprepath,Destpath,Sourceprepath,Dirname)|
compress-dirs-by-attr-name(Dirname),file-prepath(Destprepath),
file-name(Destpath),source-dir-prepath(Sourceprepath),
source-dir-name(Dirname).
zip(Destprepath,Destpath,Sourceprepath,Dirname)|
compress-dirs-by-attr-name(Dirname),file-prepath(Destprepath),
file-name(Destpath),source-dir-prepath(Sourceprepath),
source-dir-name(Dirname).

## compress all subdirectories in directories arg1
compress(Prepath,Dirname)|
compress-dirs-by-loc-dir(Dirname),source-dir-prepath(Prepath),
source-dir-name(Dirname).
gtar(Destprepath,Destpath,Sourceprepath,Dirname)|
compress-dirs-by-loc-dir(Dirname),file-prepath(Destprepath),
file-name(Destpath),source-dir-prepath(Sourceprepath),
source-dir-name(Dirname).
gzip(Prepath,Dirname)|
compress-dirs-by-loc-dir(Dirname),source-dir-prepath(Prepath),
source-dir-name(Dirname).
tar(Destprepath,Destpath,Sourceprepath,Dirname)|
compress-dirs-by-loc-dir(Dirname),file-prepath(Destprepath),
file-name(Destpath),source-dir-prepath(Sourceprepath),
source-dir-name(Dirname).
zip(Destprepath,Destpath,Sourceprepath,Dirname)|
compress-dirs-by-loc-dir(Dirname),file-prepath(Destprepath),
file-name(Destpath),source-dir-prepath(Sourceprepath),
source-dir-name(Dirname).

##find out how much file space file arg1 uses
ls(Path,Filename)|
know-filespace-usage-file(Filename),file-name(Filename),
file-prepath(Path).
du(Prepath,Partition-name)|
know-filespace-usage-partition(Partition-name),
source-dir-name(Partition-name),source-dir-prepath(Prepath).

##find out how much file space is free on filesystem arg1
```

```
df(Prepath,Partition-name)|
know-filespace-free(Partition-name),
source-dir-name(Partition-name),source-dir-prepath(Prepath).

##find out if machine arg1 is alive on the network on not
ping(Machine-name,Machine-path)|
determine-machine-connected-alive(Machine-name),
machine-prepath(Machine-path).
ruptime(Machine-name)|
determine-machine-connected-alive(Machine-name).
rsh(Machine-name,Command)|
determine-machine-connected-alive(Machine-name).
rlogin(Machine-name)|
determine-machine-connected-alive(Machine-name).

##create a file named arg1 in a preexisting directory named arg2
cd(Prepath,Dirname)|
create-file(Filename,Dirname),file-name(Filename),source-dir-name(Dirname),
source-dir-prepath(Prepath).
vi(Prepath,Filename)|
create-file(Filename,Prepath),file-name(Filename),source-dir-name(Prepath).
cat(Prepath,Filename)|
create-file(Filename,Prepath),file-name(Filename),source-dir-name(Prepath).
touch(Prepath,Filename)|
create-file(Filename,Prepath),source-dir-name(Prepath),file-name(Filename).

##create a subdirectory named arg1 in a preexisting directory named arg2
mkdir(Dirname,Parent-dir-name)|
create-dir(Dirname,Parent-dir-name),source-dir-name(Parent-dir-name),
dir-name(Dirname).
cd(Prepath,Parent-dir-name)|
create-dir(Dirname,Parent-dir-name),source-dir-name(Parent-dir-name),
source-dir-prepath(Prepath),dir-name(Dirname).

##delete all files ending in arg1
rm(Prepath,Ext)|
remove-files-by-attr-name-ext(Ext),file-name(Ext),file-prepath(Prepath).

##delete all files which contain more than arg1 bytes
rm(Prepath,Name)|
remove-files-by-attr-size-gt(Numbytes),file-size(Numbytes),file-prepath(Prepath),
file-name(Name).

## copy all files ending in arg1 to a preexisting directory named arg2
cp(Dest-prepath,Dirname,Source-prepath,Ext)|
copy-files-by-attr-name-ext(Ext,Dirname),file-name(Ext),
dest-dir-prepath(Prepath),dest-dir-name(Dirname),file-prepath(Source-prepath).

##copy all files containing less than arg1 bytes to a preexisting
directory names arg2
cp(Dest-prepath,Dirname,Source-prepath,Name)|
copy-files-by-attr-size-lt(Numbytes,Dirname),filesize(Numbytes),
dest-dir-name(Dirname),dest-dir-prepath(Dest-prepath),
file-prepath(Source-prepath),file-name(Name).

## move all files ending in arg1 to a preexisting directory named arg2
```

```
mv(Dest-prepath,Dirname,Source-prepath,Ext)|
move-files-by-attr-name-ext(Ext,Dirname),file-name(Ext),
file-prepath(Source-prepath),dest-dir-name(Dirname),
dest-dir-prepath(Dest-prepath).
copy-files-by-attr-name-ext(Ext,Dirname)|
move-files-by-attr-name-ext(Ext,Dirname),file-name(Ext),dest-dir-name(Dirname).
remove-files-by-attr-name-ext(Ext)|
move-files-by-attr-name-ext(Ext,Dirname),file-name(Ext),dest-dir-name(Dirname).

## move all files beginning with arg1 to a preexisting directory named arg2
mv(Dest-prepath,Dirname,Source-prepath,Stem)|
move-files-by-attr-name-stem(Stem,Dirname),file-name(Stem),
dest-dir-name(Dirname),file-prepath(Source-prepath),
dest-dir-prepath(Dest-prepath).
cp(Dest-prepath,Dirname,Source-prepath,Stem)|
move-files-by-attr-name-stem(Stem,Dirname),file-name(Stem),
dest-dir-name(Dirname),file-prepath(Source-prepath),
dest-dir-prepath(Dest-prepath).
rm(Source-prepath,Stem)|
move-files-by-attr-name-stem(Stem,Dirname),file-name(Stem),
dest-dir-name(Dirname),file-prepath(Source-prepath).

## move all files containing less than arg1 bytes to a preexisting directory
mv(Dest-prepath,Dirname,Source-prepath,Name)|
move-files-by-attr-size-lt(Num-bytes,Dirname),file-size(Num-bytes),
dest-dir-name(Dirname),file-name(Name),file-prepath(Source-prepath),
dest-dir-prepath(Dest-prepath).
cp(Dest-prepath,Dirname,Source-prepath,Name)|
move-files-by-attr-size-lt(Num-bytes,Dirname),file-size(Num-bytes),
dest-dir-name(Dirname),file-name(Name),file-prepath(Source-prepath),
dest-dir-prepath(Dest-prepath).
rm(Source-prepath,Name)|
move-files-by-attr-size-lt(Num-bytes,Dirname),file-size(Num-bytes),
dest-dir-name(Dirname),file-name(Name),file-prepath(Source-prepath).
```

Bibliography

Ahmadi, B., Kersting, K., & Natarajan, S. (2012). Lifted online training of relational models with stochastic gradient methods. In *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (1)(ECML/PKDD)*, pp. 585–600.

Albrecht, D. W., Zukerman, I., & Nicholson, A. E. (1998). Bayesian models for keyhole plan recognition in an adventure game. In *User Modeling and User-Adapted Interaction*, pp. 5–47.

Basu, S., Mooney, R. J., Pasupuleti, K. V., & Ghosh, J. (2001). Evaluating the novelty of text-mined rules using lexical knowledge. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2001)*, pp. 233–239, San Francisco, CA.

Berant, J., Dagan, I., & Goldberger, J. (2011). Global learning of typed entailment rules. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACl-HLT 2011)*, pp. 610–619.

Blaylock, N., & Allen, J. (2005a). Generating artificial corpora for plan recognition. In *International Conference on User Modeling (UM 2005)*. Springer.

Blaylock, N., & Allen, J. (2005b). Recognizing instantiated goals using statistical methods. In *G. Kaminka (Ed.), Workshop on Modeling Others from Observations (MOO 2005)*, pp. 79–86.

Blaylock, N., & Allen, J. F. (2004). Statistical goal parameter recognition. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pp. 297–305.

Blaylock, N. J. (2005). *Towards Tractable Agent-based Dialogue*. Ph.D. thesis, Department of Computer Science, University of Rochester, Rochester, NY.

Braz, R. D. S., Amir, E., & Roth, D. (2005). Lifted first-order probabilistic inference. In *Proceedings of Ninteenth International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pp. 1319–1325. Morgan Kaufmann.

Bui, H. H. (2003). A general model for online probabilistic plan recognition. In *Proceedings of the Eighteenth International Joint Conference on Artificial intelligence (IJCAI 2003)*.

Bui, H. H., Venkatesh, S., & West, G. (2002). Policy recognition in abstract hidden Markov model. *Journal of Artificial Intelligence Research*, *17*, 451–499.

Bunescu, R. C., & Mooney, R. J. (2007). Learning to extract relations from the web using minimal supervision. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL 2007)*, Prague, Czech Republic.

Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Jr., E. H., & Mitchell, T. (2010). Toward an architecture for never-ending language learning. In *Proceedings of*

the Conference on Artificial Intelligence (AAAI 2010), pp. 1306–1313. AAAI Press.

Carvalho, A. M., Roos, T., Oliveira, A. L., & Myllymäki, P. (2011). Discriminative learning of Bayesian networks via factorized conditional log-likelihood. *Journal of Machine Learning Research*, *12*, 2181–2210.

Charniak, E., & Goldman, R. (1991). A probabilistic model of plan recognition. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI 1991)*, pp. 160–165, Anaheim, CA.

Charniak, E., & Goldman, R. P. (1989). A semantics for probabilistic quantifier-free first-order languages, with particular application to story understanding. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI 1989)*, Detroit, MI.

Charniak, E., & McDermott, D. (1985). *Introduction to Artificial Intelligence*. Addison-Wesley, Reading, MA.

Chen, J., Muggleton, S., & Santos, J. (2008). Learning probabilistic logic models from probabilistic examples. *Machine Learning*, *73*(1), 55–85.

Cowie, J., & Lehnert, W. (1996). Information extraction. *CACM*, *39*(1), 80–91.

De Raedt, L., & Dehaspe, L. (1997). Clausal discovery. *Machine Learning*, *26*(2–3), 99–146.

De Raedt, L., & Kersting, K. (2004). Probabilistic inductive logic programming. In *Algorithmic Learning Theory*, pp. 19–36.

den Broeck, G. V., Taghipour, N., Meert, W., Davis, J., & De Raedt, L. (2011). Lifted probabilistic inference by first-order knowledge compilation. In *Proceedings of the Twentieth International Joint Conference on Artificial intelligence (IJCAI 2011)*, pp. 2178–2185. IJCAI/AAAI.

Dinh, Q.-T., Exbrayat, M., & Vrain, C. (2011). Generative structure learning for Markov logic networks based on graph of predicates. In *Proceedings of the 22nd international joint conference on Artificial Intelligence - Volume Volume Two (IJCAI 2011)*, IJCAI 2011, pp. 1249–1254. AAAI Press.

Domingos, P., & Lowd, D. (2009). *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool, San Rafael, CA.

Doppa, J. R., NasrEsfahani, M., Sorower, M. S., Dietterich, T. G., Fern, X., & Tadepalli, P. (2010). Towards learning rules from natural texts. In *Proceedings of the NAACL HLT 2010 First International Workshop on Formalisms and Methodology for Learning by Reading (FAM-LbR 2010)*, pp. 70–77, Stroudsburg, PA, USA. Association for Computational Linguistics.

Elvira-Consortium (2002). Elvira: An environment for probabilistic graphical models. In *Proceedings of the Workshop on Probabilistic Graphical Models*, Cuenca, Spain.

Feldman, R., & Dagan, I. (1995). Knowledge discovery in textual databases (kdt). In *In Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD 1995)*, pp. 112–117. AAAI Press.

Fellbaum, C. (Ed.). (1998). *WordNet An Electronic Lexical Database*. The MIT Press, Cambridge, MA ; London.

Florian, R., Hassan, H., Ittycheriah, A., Jing, H., Kambhatla, N., Luo, X., Nicolov, N., & Roukos, S. (2004). A statistical model for multilingual entity detection and tracking. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT 2004)*, pp. 1–8.

Friedman, N., Getoor, L., Koller, D., & Pfeffer, A. (1999). Learning probabilistic relational models. In *Proceedings of the Sixteenth International Joint Conference on Artificial intelligence (IJCAI 1999)*, Stockholm, Sweden.

Garrette, D., Erk, K., & Mooney, R. (2011). Integrating logical representations with probabilistic information using Markov logic. In *Proceedings of the International Conference on Computational Semantics (ICCS 2011)*, pp. 105–114, Oxford, England.

Getoor, L., & Taskar, B. (Eds.). (2007). *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, MA.

Gogate, V., & Dechter, R. (2007). Samplesearch: A scheme that searches for consistent samples. In *Proceedings of Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS 2007)*.

Gogate, V., & Domingos, P. (2011). Probabilistic theorem proving. In *Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty*

in Artificial Intelligence (UAI 2011), pp. 256–265, Corvallis, Oregon. AUAI Press.

Greiner, R., Su, X., Shen, B., & Zhou, W. (2002). Structural extension to logistic regression: Discriminative parameter learning of belief net classifiers. In *Proceedings of the Eighteenth Annual National Conference on Artificial Intelligence (AAAI 2002)*, pp. 167–173.

Grice, H. P. (1975). Logic and conversation. In Cole, P., & Morgan, J. L. (Eds.), *Syntax and Semantics: Vol. 3: Speech Acts*, pp. 41–58. Academic Press, San Diego, CA.

Han, J., & Fu, Y. (1995). Discovery of multiple-level association rules from large databases. In *In Proceedings of 1995 International Conference on Very Large Data Bases (VLDB 1995)*, pp. 420–431.

Heckerman, D., & Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. In *Machine Learning*, pp. 20–197.

Hong, J. (2001). Goal recognition through goal graph analysis. *Journal of Artificial Intelligence Research*, *15*, 1–30.

Horvitz, E., & Paek, T. (1999). A computational architecture for conversation. In *Proceedings of the Seventh International Conference on User Modeling*, pp. 201–210. Springer.

Huber, M. J., Durfee, E. H., & Wellman, M. P. (1994). The automated mapping of plans for plan recognition. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence (UAI 1994)*, pp. 344–351. Morgan Kaufmann.

Huynh, T. N., & Mooney, R. J. (2008). Discriminative structure and parameter learning for Markov logic networks. In *In Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, pp. 416–423, Helsinki, Finland.

Kakas, A. C., Kowalski, R. A., & Toni, F. (1993). Abductive logic programming. *Journal of Logic and Computation*, *2*(6), 719–770.

Kaminka, G. A., Pynadath, D. V., & Tambe, M. (2002). Monitoring teams by overhearing: A mulit-agent plan-recognition approach. *Journal of Artificial Intelligence Research*, *17*, 83–135.

Kate, R. J., & Mooney, R. J. (2009). Probabilistic abduction using Markov logic networks. In *Proceedings of the IJCAI-09 Workshop on Plan, Activity, and Intent Recognition*, Pasadena, CA.

Kautz, H. A. (1987). *A Formal Theory of Plan Recognition*. Ph.D. thesis, Department of Computer Science, University of Rochester, Rochester, NY. Technical Report 215.

Kautz, H. A., & Allen, J. F. (1986). Generalized plan recognition. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI 1986)*, pp. 32–37, Philadelphia, PA.

Kersting, K., & De Raedt, L. (2007). Bayesian Logic Programming: Theory and tool. In Getoor, L., & Taskar, B. (Eds.), *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, MA.

Kersting, K., & De Raedt, L. (2001). Towards combining inductive logic programming with Bayesian networks. In *Proceedings of the Eleventh International Conference on Inductive Logic Programming (ILP 2001)*, pp. 118–131, Strasbourg, France.

Kersting, K., & De Raedt, L. (2008). Basic principles of learning Bayesian logic programs. In *Probabilistic Inductive Logic Programming*, pp. 189–221.

Kimmig, A., Santos Costa, V., Rocha, R., Demoen, B., & De Raedt, L. (2008). On the efficient execution of problog programs. In *Proceedings of the 24th International Conference on Logic Programming (ICLP 2008)*, pp. 175–189, Berlin, Heidelberg. Springer-Verlag.

Koivisto, M., & Sood, K. (2004). Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research*, *5*, 549–573.

Koller, D., & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Kulesza, A., & Pereira, F. (2008). Structured learning with approximate inference. In *Advances in neural information processing systems 20*.

Lauritzen, S. L. (1995). The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, *19*, 191–201.

Lavrač, N., & Džeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood.

Lesh, N., & Etzioni, O. (1995). A sound and fast goal recognizer. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 1995)*.

Levesque, H. J. (1989). A knowledge-level account of abduction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI 1989)*, pp. 1061–1067, Detroit, MI.

Lin, D., & Pantel, P. (2001). Discovery of inference rules for question answering. *Natural Language Engineering*, *7*(4), 343–360.

Lin, D. (1998). An information-theoretic definition of similarity. In *In Proceedings of the 15th International Conference on Machine Learning (ICML 1998)*, pp. 296–304. Morgan Kaufmann.

McCreath, E., & Sharma, A. (1998). Lime: A system for learning relations. In *Ninth International Workshop on Algorithmic Learning Theory*, pp. 336–374. Springer-Verlag.

Milch, B., Zettlemoyer, L. S., Kersting, K., Haimes, M., & Kaelbling, L. P. (2008). Lifted probabilistic inference with counting formulas. In *Proceedings of the Twenty Third National Conference on Artificial Intelligence (AAAI 08)*, AAAI 08, pp. 1062–1068. AAAI Press.

Muggleton, S. (2000). Learning stochastic logic programs. In *Proceedings of the AAAI2000 Workshop on Learning Statistical Models from Relational Data*.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, *13*, 245–286.

Muggleton, S. (2003). Learning structure and parameters of stochastic logic programs. In *Proceedings of the twelfth international conference on Inductive logic programming (ILP 2002)*, pp. 198–206, Berlin, Heidelberg. Springer-Verlag.

Muggleton, S. H. (Ed.). (1992). *Inductive Logic Programming*. Academic Press, New York, NY.

Nahm, U. Y., & Mooney, R. J. (2000). A mutually beneficial integration of data mining and information extraction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*, pp. 627–632, Austin, TX.

Nau, D., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., & Yaman, F. (2003). Shop2: An HTN planning system. *Journal of Artificial Intelligence Research*, *20*, 379–404.

Ng, H. T., & Mooney, R. J. (1990). The role of coherence in abductive explanation. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pp. 337–442, Detroit, MI.

Ng, H. T., & Mooney, R. J. (1992). Abductive plan recognition and diagnosis: A comprehensive empirical evaluation. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR 1992)*, pp. 499–508, Cambridge, MA.

Nijssen, S., & Kok, J. N. (2003). Efficient frequent query discovery in FARMER. In *Proceedings of the Seventh Conference in Principles and Practices of Knowledge Discovery in Database (PKDD 2003)*, pp. 350–362. Springer.

Nilsson, D. (1998). An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. *Statistics and Computing*, *8*, 159–173.

Niu, F., Zhang, C., Re, C., & Shavlik, J. (2012). Scaling inference for markov logic via dual decomposition. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*.

Park, S., & Kautz, H. (2008). Hierarchical recognition of activities of daily living using multi-scale, multiperspective vision and rfid. In *The fourth IET International Conference on Intelligent Environments*.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. MKP, CA.

Pedersen, T., Patwardhan, S., & Michelizzi, J. (2004). Wordnet:: Similarity - measuring the relatedness of concepts. In *Nineteenth National Conference on Artificial Intelligence (AAAI 2004)*, pp. 1024–1025.

Peirce, C. S. (1958). *Collected Papers of Charles Sanders Peirce*. MIT Press, Cambridge, Mass.

Perlich, C., & Merugu, S. (2005). Multi-relational learning for genetic data: Issues and challenges. In Džeroski, S., & Blockeel, H. (Eds.), *Fourth International Workshop on Multi-Relational Data Mining (MRDM 2005)*.

Poole, D. (1993). Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, *64*, 81–129.

Pople, H. E. (1973). On the mechanization of abductive logic. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI 1973)*, pp. 147–152.

Pynadath, D. V., & Wellman, M. P. (2000). Probabilistic state-dependent grammars for plan recognition. In *In Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI 2000)*, pp. 507–514. Morgan Kaufmann Publishers.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo,CA.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, *5*(3), 239–266.

Quinlan, J. R., & Cameron-Jones, R. M. (1993). FOIL: A midterm report. In *Proceedings of the European Conference on Machine Learning*, pp. 3–20, Vienna.

Raghavan, S., & Mooney, R. J. (2011). Abductive plan recognition by extending Bayesian logic programs. In *Proceedings of the European Conference on Machine Learning/Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD 2011)*, Vol. 2, pp. 629–644.

Raghavan, S., Mooney, R. J., & Ku, H. (2012). Learning to read between the lines using Bayesian Logic Programs. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL 2012)*.

Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. In *In Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 1995)*, pp. 448–453.

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning Journal*, *62*, 107–136.

Rosner, B. (2005). *Fundamentals of Biostatistics*. Duxbury Press.

Russell, S., Binder, J., Koller, D., & Kanazawa, K. (1995). Local learning in probabilistic networks with hidden variables. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 1995)*, pp. 1146–1152, Montreal, Canada.

Russell, S., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (2 edition). Prentice Hall, Upper Saddle River, NJ.

Sarawagi, S. (2008). Information extraction. *Foundations and Trends in Databases*, *1*(3), 261–377.

Saria, S., & Mahadevan, S. (2004). Probabilistic plan recognition in multiagent systems. In *International Conference on Automated Planning and Scheduling (ICAPS 2004)*.

Sato, T. (1995). A statistical learning method for logic programs with distribution semantics. In *Proceedings of the twelfth international conference on logic programming (ICLP 1995)*, pp. 715–729. MIT Press.

Schoenmackers, S., Etzioni, O., & Weld, D. S. (2008). Scaling textual inference to the web. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2008)*, pp. 79–88, Stroudsburg, PA, USA. Association for Computational Linguistics.

Schoenmackers, S., Etzioni, O., Weld, D. S., & Davis, J. (2010). Learning first-order Horn clauses from web text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2010)*, pp. 1088–1098, Stroudsburg, PA, USA. Association for Computational Linguistics.

Shapiro, E. Y. (1983). *Algorithmic Program Debugging*. MIT Press, Cambridge, MA.

Silander, T., & Myllymäki, P. (2006). A simple approach for finding the globally optimal Bayesian network structure. In *Proceedings of Uncertainty in Artificial Intelligence (UAI 2006)*.

Singla, P., & Domingos, P. (2008). Lifted first-order belief propagation. In *Proceedings of the Twenty Third National Conference on Artificial Intelligence (AAAI 08)*.

Singla, P., & Mooney, R. (2011). Abductive Markov Logic for plan recognition. In *Twenty-fifth National Conference on Artificial Intelligence (AAAI 2011)*.

Snow, R., O'Connor, B., Jurafsky, D., & Ng, A. Y. (2008). Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2008)*, EMNLP 2008, pp. 254–263. Association for Computational Linguistics.

Sorower, M. S., Dietterich, T. G., Doppa, J. R., Walker, O., Tadepalli, P., & Fern, X. (2011). Inverting Grice's maxims to learn rules from natural language extractions. In *Proceedings of Advances in Neural Information Processing Systems 24 (NIPS 2011)*.

Srinivasan, A. (2001). *The Aleph manual*. `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/`.

Stickel, M. E. (1988). A Prolog-like inference system for computing minimum-cost abductive explanations in natural-language interpretation. Tech. rep. Technical Note 451, SRI International, Menlo Park, CA.

Tamaddoni-Nezhad, A., Chaleil, R., Kakas, A., & Muggleton, S. (2006). Application of abductive ilp to learning metabolic network inhibition from temporal data. *Machine Learning*, *64*(1-3), 209–230.

Teyssier, M., & Koller, D. (2005). Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *Proceedings of Uncertainty in Artificial Intelligence (UAI 2005)*, pp. 584–590.

Thompson, C., & Mooney, R. (1994). Inductive learning for abductive diagnosis. In *Twelfth National Conference on Artificial Intelligence (AAAI 1994)*.

Wainwright, M. J. (2006). Estimating the wrong graphical model: Benefits in the computation-limited setting. *Journal Machine Learning Research*, *7*, 1829–1859.

Welty, C., Barker, K., Aroyo, L., & Arora, S. (2012). Query driven hypothesis generation for answering queries over nlp graphs. In *International Semantics Web Conference (2) (ISWC)*, pp. 228–242.

Wu, Z., & Palmer, M. (1994). Verb semantics and lexical selection. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics (ACL 1994)*, pp. 133–138.

Yates, A., & Etzioni, O. (2007). Unsupervised resolution of objects and relations on the web. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT 2007)*.

Zeichner, N., Berant, J., & Dagan, I. (2012). Crowdsourcing inference-rule evaluation.. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL 2012)*, pp. 156–160. The Association for Computer Linguistics.

Vita

Sindhu Vijaya Raghavan was born in 1981 in Bangalore, India. She finished high school at Vijaya High School and National College, both located in Bangalore in 1999. She then went on to study Computer Science and Engineering at the R.V. College of Engineering in the Visvesvaraya Technological University, where she received a Bachelor of Engineering degree and was awarded the fifth rank for the university in 2003. She then went to work as a software engineer in Intel, I2 technologies, and Delmia Solutions Private Ltd. After her brief stint of four years in the software industry, she returned to graduate school at the University of Texas at Austin to pursue a doctorate degree in Computer Science in 2007. In 2010, she received her Master of Science degree in Computer Science from the University of Texas at Austin. Since then, she has continued her research in the areas of machine learning and natural language processing at the University of Texas at Austin. After graduation, she will join Samsung Telecommunications in Dallas starting January 2013.

Permanent address: sindhu.vijayaraghavan@gmail.com

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.